Admission Control and Dynamic Adaptation for a Proportional-Delay DiffServ-Enabled Web Server

Lee Cheuk Man

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

November, 2002

Abstract

We consider a web server that can provide differentiated services to clients with different quality of service (QoS) requirements. The web server can provide $N \geq 1$ classes of proportional-delay differentiated services to heterogeneous clients. An operator can specify *fixed* performance spacings between classes, namely, $r_{i,i+1} > 1$, for $i = 1, \ldots, N-1$. Requests in class $i+1$ are guaranteed to have an average waiting time which is $1/r_{i,i+1}$ of the average waiting time of class i requests. With PDDS, we can provide consistent performance spacings over a wide range of system loading and this simplifies many pricing issues. In addition, each client can specify a maximum average waiting time requirement to be guaranteed by the PDDS-enabled web server. We show that, in general, the problem of assigning clients to service classes in order to optimize system efficacy is NP-complete. We propose two efficient admission control algorithms so that a web server can provide the QoS guarantees and, at the same time, classify each client to its "lowest" admissible class, resulting in lowest usage cost for the admitted client. We also consider how to perform end-point dynamic adaptation such that admitted clients can submit requests at a lower class and further reduce their usage costs, without violating their QoS requirements. We propose two dynamic adaptation algorithms: one is server-based and the other is client-based. The client-based adaptation is distributed and is based on a non-cooperative game technique. We carry out experiments to illustrate the effectiveness of these algorithms under different utility functions and traffic arrival patterns (e.g., Poisson, MMPP, and Pareto). We report

extensive experimental results to illustrate the effectiveness of our proposed algorithms.

摘要

我們考慮一個可以提供「分級服務」予有不同「服務品質」要求的顧客的萬維網服務器。這個萬維網服務器可提供多過一個級別的「比例延遲分級服務」給不同的顧客。服務器能夠指明級別間的固定表現比例R，以保證i＋1級的要求之等候時間是i級的要求之等候時間的R份之一。「比例延遲分級服務」能夠於不同的系統負荷中提供穩定的表現，從而簡化標價事項。同時，提供「比例延遲分級服務」的服務器能保證顧客的最大的平均等候時間要求。我們會證明分配顧客到不同服務級別的問題是「NP完全」的。我們建議兩個有效的容許控制算法使萬維網服務器能夠把顧客分類到最低的級別，使顧客可享受服務品質，也同時只需支付最低的支出。我們也同時考慮怎樣去進行端點動態適應使接納的顧客可於較低級別提交要求從而減少支出。我們提供兩個動態適應算法：服務器式和顧客式。顧客式適應是分散的和建基於博奕論。我們用實驗去說明這些算法於不同的到達模式（泊松、馬爾可夫鏈）和功用函數的實際性。我們也提供詳細的實驗去說明這些算法的效率。

Acknowledgments

I wish to express my gratitude to my supervisor Dr. John C. S. LUI for giving continuous support and guidance in research. I would like to thank Dr. David K. Y. YAU for giving many helpful advice. I would also like to thank Dr. Leizhen CAI and Lapchi LAU for their support in the proof of NP-completeness of the optimization problem.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet is becoming more commercially oriented and businesses are now using web servers to disseminate information. Therefore, the effect of access latency at web servers has become more important. Conventional web servers use a single class approach in serving client requests. This does not provide adequate performance when different clients may have different QoS requirements and are willing to pay different prices to attain their desired QoS. Hence, there is a need to support *multiple* classes of service at a web server, in order to extend network level service differentiation (e.g., the DiffServ model) to true end-to-end *application level* service differentiation.

There are several ways for a web server to provide differentiated services. For example, a strict priority policy can be used, in which clients submit requests in different priority classes, and the web server always serves the next request from the highest priority class that is backlogged. Some drawbacks of the strict priority policy are (a) the possibility of starvation for requests in the lower priority classes, and (b) the performance spacings between different classes are *load dependent*, introducing pricing complication. For example, if a client X is charged at a rate of R_1 and another client Y is charged at a rate of R_2, where $R_2 > R_1$, then Y should expect its performance to be *proportionately* better than that of X (i.e., the performance of Y is R_2/R_1 that of X), regardless of system loading. This type of performance guarantees

cannot be easily achieved with a strict priority policy.

We target a web server which can provide a differentiated service that has the following properties:

- *Consistency:* service differentiation is consistent (i.e., higher classes receive better service) and the performance differentiation is *independent* of variations in class load.

- *Controllability:* the operator of the web server can specify and control the performance spacings between offered classes of service, according to the pricing structure.

In [1], the authors propose an Internet service model called proportional-delay differentiated services, which has the above mentioned consistency and controllability properties. In the service model, the performance spacing between class $i+1$ and class i can be specified as a *fixed* ratio $r_{i,i+1}$. If this ratio can be maintained over a wide range of system loading, then a user of class $i+1$, who is paying at a rate $r_{i,i+1}$ higher than a user of class i, will consistently have a performance that is $r_{i,i+1}$ better than the class i user. To realize proportional-delay differentiated services, the authors in [1] propose to use the *time-dependent priority* (TDP) service discipline in [2]. In [3, 4], the authors illustrate the necessary and sufficient conditions under which the controllability and consistency properties can be maintained.

In this thesis, we consider proportional-delay differentiated services at a web server, say $\mathcal{S}$. $\mathcal{S}$ provides waiting time differentiation for $N > 1$ classes of requests. Let W_i be the expected waiting time of class i requests, for $i = 1, \ldots, N$. The operator of the web server $\mathcal{S}$ specifies a fixed performance spacing $r_{i,i+1} > 1$ such that

$$W_i/W_{i+1} = r_{i,i+1} \qquad \text{for } i = 1, 2, \ldots, N-1.$$

For example, if $r_{i,i+1} = 1.5$, then the operator can legitimately charge class $i+1$ clients a usage rate 50% higher than that of class i clients. In addition to

this performance spacing guarantee, each client specifies a maximum average waiting time for its requests to be guaranteed by $\mathcal{S}$. We consider the following technical issues:

- Efficient admission control so that $\mathcal{S}$ can provide the requested performance differentiation and guarantees.
- Efficient assignment or mapping of client requests into the service classes, so that an admitted client's performance requirement can be satisfied.
- Dynamic adaptation such that, depending on the server workload, a client can assign requests to a lower service class (i.e., lower than the class which was initially prescribed at admission control time) and can still receive service consistent with its performance requirement. This way, a client can pay a lower usage cost while still obtaining satisfactory service.

The structure of the thesis is organized as follows. In Chapter 2, we provide the necessary background of proportional-delay differentiated services. We also formulate the problem of admission control, client classification and dynamic adaptation. We show that, in general, the client classification problem to optimize system efficacy is NP-complete. In Chapter 3, we present two efficient admission control algorithms and state their important properties. In Chapter 4, we present two adaptation algorithms: One is server-based (i.e., a centralized algorithm) while the other is client-based (i.e., a distributed algorithm). The client-based algorithm is based on a *non-cooperative* game approach and has a low computational complexity. In Chapter 5, we present experimental results to illustrate the effectiveness of the proposed algorithms. Related work is presented in Chapter 6.

Chapter 2

Background & Problem Formulation

Let us present the background of proportional-delay differentiated services (PDDS) [1, 3, 4]. Under PDDS, there are $N > 1$ service classes such that class $i+1$ requests will receive better performance compared with class i requests, for $i = 1, \ldots, N-1$. In this work, we consider performance as the average waiting time of a client's requests. The waiting time of a request is the time the request spends in the server's queue before it receives service. Let W_i be the achieved long-term average waiting time of class i requests. A PDDS web server tries to guarantee that the ratio of the achieved long-term average waiting time between classes i and $i+1$ is equal to a *fixed* and *prespecified* ratio, $r_{i,i+1}$, where

$$W_i/W_{i+1} = r_{i,i+1} \qquad \text{for } i = 1, \ldots, N-1 \tag{2.1}$$

The objective is to maintain $r_{i,i+1} > 1$ across a wide range of system loading. As mentioned, PDDS can be achieved using the time-dependent priority (TDP) scheduler [1]. In general, TDP is a *non-preemptive* priority scheduling algorithm with a set of control variables $b_i, 1 \leq i \leq N$, where $0 \leq b_1 \leq b_2 \leq \cdots \leq b_N$. Specifically, if the k-th request of class i arrives at the system at time τ_k, then its *instantaneous* priority at time t (for $t \geq \tau_k$),

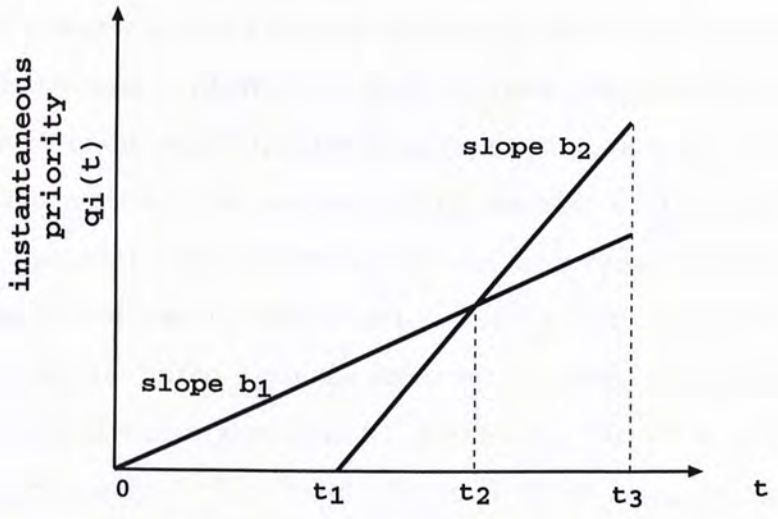


Figure 2.1: A two-class TDP where $b_1 < b_2$.

denoted by $q_i^k(t)$, is

$$q_i^k(t) = (t - \tau_k)b_i \ . \tag{2.2}$$

To clearly illustrate the concept, we use a two-class TDP as depicted in Figure 2.1. Assume that the first request of class 1 arrives at time 0 and the first request of class 2 arrives at time t_1. Both requests remain in the system until time t_3. During the time interval $(t_1, t_2]$, the class 1 request will have a higher priority than the class 2 request. But since the control parameter b_2 is larger than b_1, after time $t > t_2$, the class 2 request will have a higher priority. Because of this property, requests in higher classes cannot monopolize the system resources and cause the starvation problem.

Let $N_i(t)$ denote the number of class i requests waiting in the queue at time t and $q_i(t)$ the priority of the request at the head of the class i queue. When a web server $\mathcal{S}$ is ready to service a request at time t, it chooses a request from class i^* where

$$i^*(t) = arg \max_{i=1..N, N_i(t)>0} \{q_i(t)\} . \tag{2.3}$$

Ties for the highest priority are broken by serving the request that has been waiting the longest in the system. If there is no request in the system, the server is idle and will be activated by any newly arriving request. Note that for

the TDP scheduler, a class i request increases in priority at a faster rate than requests of any class j, where $j < i$. In [3, 4], the authors derive the *necessary* and *sufficient* conditions for feasible delay ratios (Equation (2.1)). Specifically, for a two-class system, if the system loading ρ satisfies $1 - 1/r_{1,2} < \rho < 1$, then by setting the control parameters $b_1 = 1$ and $b_2 = \rho/(\rho - 1 + \frac{1}{r_{1,2}})$, one can achieve the desired waiting time spacing. For a system with more than two classes of traffic, the authors give the necessary conditions for feasible spacings, and an efficient iterative algorithm for determining the values of the control parameters b_i, $i = 1, \ldots, N$. Please refer to [3, 4] for a detailed derivation of these parameter values.

Consider the utility of a PDDS-enabled web server offering, say, video-on-demand service. In this case, a class i client who wants to access a video will experience a smaller start-up latency than a client in class $i - 1$. In exchange, the class i client will be charged at a higher usage rate than the class $i - 1$ client. Our focus is on providing *fundamental understanding* for the design of such a PDDS-enabled web server.

Assume that there are $M > 0$ potential clients requesting service from a PDDS-enabled web server $\mathcal{S}$. Each of these clients can be viewed as an aggregation of many individual users (e.g., users from the same company or the same network domain). A client, say j, specifies two parameters for its desired QoS:

- λ_j^{max}: j's maximum offered traffic rate to the server.
- W_j^{max}: the maximum average waiting time for client j's requests before service is obtained.

If a client is admitted to the system and is assigned to class i, the client is charged an admission cost of A_i, where $A_1 \leq A_2 \leq \cdots \leq A_N$. $\mathcal{S}$ also charges a usage cost of ϕ_i for each request in class i, where $\phi_1 \leq \phi_2 \leq \cdots \leq \phi_N$.

The problems we want to address are:

1. *Admission control and class assignment:*

   Given the workload (λ_j^{max}) and the QoS requirement (W_j^{max}) of client j requesting service, should $\mathcal{S}$ admit this client, such that the QoS requirements of all the admitted clients will be satisfied? Also, when a system decides to admit client j, what is the *lowest* possible class assignment for j, such that j will pay the lowest possible usage cost?

2. *Dynamic class adaptation:*

   For those admitted clients, their request arrival rates may be less than their specified maximum request arrival rates. Therefore, rather than use the assigned class obtained during the admission control process, a client may choose to submit requests at a lower class. This way, the client may enjoy its desired level of service at a reduced usage cost. We consider the problem of how each client can adapt to the traffic loading at a web server $\mathcal{S}$ and adjust its service class dynamically. The main challenge is to guarantee that we will not violate the maximum average request waiting time required by the client.

Before we proceed, let us define the following notation. Let M' be the number of admitted clients to $\mathcal{S}$. In general, we have $M' \leq M$. The admitted class vector, denoted by $\boldsymbol{C}^a = [C_1^a, C_2^a, \ldots, C_{M'}^a]$, represents the class assignment of each admitted client after the admission control and class assignment process[1] . The class assignment for client i is $C_i^a \in \{1, 2, \ldots, N\}$, for $i = 1, \ldots, M'$. After the admission control, an admitted client may dynamically adapt to the loading at $\mathcal{S}$ and lower its assigned class. The class vector at time $t > 0$ for all admitted clients is denoted by $\boldsymbol{C}(t) = [C_1, C_2, \ldots, C_{M'}]$ where $C_i \in \{1, \ldots, N\}$ is the class chosen by client i. It is easy to observe that $\boldsymbol{C}(0) = \boldsymbol{C}^a$ and $\boldsymbol{C}(t) \leq \boldsymbol{C}^a$ for $t > 0$. The total maximum arrival rate of class

[1] In this thesis, we assume that the system will assign a class value of 0 to those clients that the system cannot admit.

k, denoted as $\Lambda_{c_k}^{max}$, is: [2]

$$\Lambda_{c_k}^{max} = \sum_{j=1}^{M} \lambda_j^{max} \mathbf{1}\{C_j = k\} \quad k = 1, 2, \ldots, N.$$

Let W_{c_k} denote the average waiting time of class k requests. Based on the conservation law [2], we have:

$$\sum_{k=1}^{N} \Lambda_{c_k}^{max} W_{c_k} = \sum_{k=1}^{N} \Lambda_{c_k}^{max} W(\lambda) \tag{2.4}$$

where $W(\lambda)$ represents the average waiting time that would result if the aggregate traffic were serviced by a work-conserving FCFS server of the same capacity as $\mathcal{S}$. Define $\sigma_1 = 1$ and $\sigma_i = \sigma_{i-1}/r_{i-1,i}$ for $i = 2, \ldots, N$. Based on Equation (2.1), we have $r_{i-1,i} = W_{c_{i-1}}/W_{c_i}$. Therefore

$$\sigma_i = W_{c_i}/W_{c_1} \quad i = 1, \ldots, N.$$

Based on the above equation, we can express W_{c_i} in terms of W_{c_k} as:

$$W_{c_i} = \sigma_i \frac{W_{c_k}}{\sigma_k} \quad i = 1, 2, \ldots, N. \tag{2.5}$$

Substituting Equation (2.5) into Equation (2.4), we have

$$\frac{W_{c_k}}{\sigma_k} \sum_{i=1}^{N} \Lambda_{c_i}^{max} \sigma_i = \sum_{i=1}^{N} \Lambda_{c_i}^{max} W(\lambda).$$

After rearranging terms, we can express W_{c_k} as

$$W_{c_k} = \frac{\sigma_k \left(\sum_{j=1}^{N} \Lambda_{c_j}^{max} \right) W(\lambda)}{\sum_{j=1}^{N} \sigma_j \Lambda_{c_j}^{max}} \text{ for } k = 1, \ldots, N. \tag{2.6}$$

Let U_i be a function representing the utility of client i. Each client can have a different utility function. In this thesis, the utility function we consider has a form which is illustrated in Figure 2.2. Let S_i be the cost of client i. The net

[2] $$\mathbf{1}\{C_j = k\} = \begin{cases} 1 & \text{if } C_j = k \\ 0 & \text{otherwise} \end{cases}$$

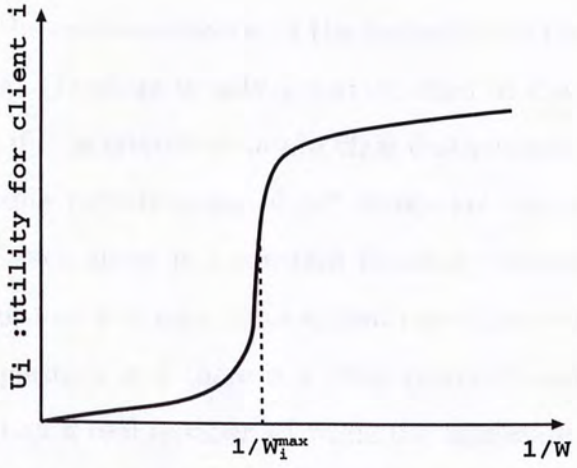


Figure 2.2: General form of client's utility function vs. inverse of the waiting time

utility of client i is then $(U_i - S_i)$. For the server, the utility is the amount of cost paid by all admitted clients, i.e. $\sum_{i=1}^{M'} S_i$. Let R be the maintenance cost of the server. We define the system efficacy V as the sum of the net utilities of all admitted clients and the web server. Our objective is to

$$\begin{aligned} \max V &= \sum_{i=1}^{M'}(U_i(W_{c_k}) - S_i) + (\sum_{i=1}^{M'} S_i - R) \\ &= \sum_{i=1}^{M'} U_i(W_{c_k}) - R \\ \text{s. t.} \quad W_{c_k} &\leq W_i^{max} \quad i = 1, \ldots, M' \text{ and } k = C_i^a \end{aligned} \tag{2.7}$$

i.e., we seek to maximize the system efficacy V under the constraint that the expected waiting time of an admitted client i is less than or equal to its QoS requirement W_i^{max}. In [5], the authors show that if a request has a utility function with the form similar to Figure 2.2, then one needs to apply admission control to maximize the system efficacy V. In the following, we show that the above optimization problem is NP-complete.

Theorem 2.1 The constructed optimization problem given in Equation (2.7) is NP-complete.

Proof: Consider the decision version of the optimization problem in Equation (2.7). Assume that (1) there is only 1 service class in the system, so that a client is in class 1 if it is admitted and in class 0 otherwise, (2) the maximum average waiting time requirements of all clients are the same, and (3) the utility function of each client is a constant function. Because of the first two restrictions, we can find the maximum arrival rate allowed in the system, say Λ^{max}. Now, the question is if there is a class vector C such that the system efficacy is larger than a real number V' while the aggregate arrival rate is less than Λ^{max}.

Given a class vector $\boldsymbol{C}$, it can be checked in linear time if the aggregate arrival rate is larger than Λ^{max}. The average waiting time can be calculated in polynomial time. The system efficacy can then be found by $\sum_{i=1}^{M'} U_i(W_{c_i})$ in polynomial time. Clearly, the whole process can be done in polynomial time and so the decision problem is in NP. Now, we want to transform the decision problem to a known NP-complete KNAPSACK problem. The arrival rate constraint is transformed to the size constraint of KNAPSACK. The summation of system efficacy is transformed to the summation of values in KNAPSACK. Clearly, the transformation can be done in polynomial time. Therefore our decision problem is in NP-complete.

Since the above decision problem is only a decision version of the optimization problem in Equation (2.7) with three restrictions, the optimization problem in Equation (2.7) must be in NP-complete. ∎

In general, finding the solution to the optimization problem in Equation (2.7) can be computationally expensive. A straightforward approach is to perform an exhaustive search. The search has a computational complexity of $\Theta((N+1)^M)$ in evaluating the expression of Equation (2.6) so as to choose the optimal configuration. Since the number of clients M can be very large, the computation cost is prohibitive even for a small number of classes N. In the

following, we propose two efficient admission control algorithms such that at the end of the admission control process, we can determine (1) the clients that the web server can admit, and (2) the lowest possible admitted class vector $\boldsymbol{C}^a = [C_1^a, C_2^a, \ldots, C_{M'}^a]$ for those admitted clients.

Chapter 3

Admission Control and Resource Provisioning

In this chapter, we explain how to perform the admission control and the class assignment for a PDDS-enabled web server.

3.1 Admission Control & Class Assignment

To subscribe service from the server $\mathcal{S}$, each client has to go through the admission control procedure. Each client j will provide the information, λ_j^{max} and W_j^{max}, to the server $\mathcal{S}$. In return, the server $\mathcal{S}$ will indicate whether it can admit client j or not. If the system can admit client j, it will also notify client j of the assigned class index, $C_j^a \in \{1, \ldots, N\}$. As long as client j marks all its requests to $\mathcal{S}$ in class C_j^a, the server $\mathcal{S}$ can *guarantee* that the long term average waiting time for client j is less than or equal to the QoS requirement W_j^{max}. We propose the following two admission control/class assignment algorithms.

3.2 Maximum Profit Algorithm (MPA)

The first algorithm is MPA. The objective is to admit a client having a more stringent maximum average waiting time requirement first. The rationale is

that if there are two clients i and j with requirements of W_i^{max} and W_j^{max}, respectively, and $W_i^{max} < W_j^{max}$, it is reasonable to assume client i is willing to pay a higher usage cost than client j so as to receive better service. By admitting client i, the service provider may obtain a higher profit. The MPA algorithm is given as:

MPA Admission Control

1. Sort the maximum average waiting time requirement of all clients from smallest to largest. After the sorting, we assume client 1 (respectively, client M) has the smallest (respectively, largest) maximum average waiting time requirement.
2. Let Ω be the set of all admitted clients. Initialize $\Omega = \emptyset$ and initialize the admitted class vector $\boldsymbol{C}^a = \mathbf{0}$;
3. **for** ($i = 1$ **to** M) {`/* test all` M `clients */`
4. $\Omega' = \Omega \bigcup$ client i; $\boldsymbol{C}^a_{temp} = \boldsymbol{C}^a$; satisfied_flag = false;
5. assign client i in class 1;
6. **while** (satisfied_flag == false) {
7. compute delay of all clients in Ω' based on Eq.(2.6);
8. **if** (waiting times of all clients in Ω' are satisfied) {
9. $\Omega = \Omega'$; satisfied_flag=true; }
10. **else**{`/* perform class upgrade for unsatisfied clients*/`
11. **if** (there is any unsatisfied client with class equal N)
`/*cannot admit client` i`, restore` $\boldsymbol{C}^a$`*/`
12. $\boldsymbol{C}^a = \boldsymbol{C}^a_{temp}$; satisfied_flag = true;
13. **else** `/* upgrade unsatisfied clients */`
14. increase the class of all unsatisfied clients in Ω' by 1;
15. }
16. } `/* termination of while loop */`
17. } `/* termination of for loop */`

18. **return** (Ω and $\boldsymbol{C}^a$);

Let us explain the rationale of the MPA algorithm. Under MPA admission control, we test whether we can admit a tagged client (line 3). For this tagged client i, we first assign it to class one (line 5). By adding this client i, we may change the waiting times of previously admitted clients. We test whether this new additional client will violate the QoS of other clients in Ω' (line 7). If the addition does not violate the QoS of any client, we can admit this tagged client i (line 9). On the other hand, if there is any QoS violation and the unsatisfied clients are already in class N, this implies that we cannot admit the tagged client i (line 11-12). If there is QoS violation and none of the unsatisfied clients is in class N, we can upgrade all the unsatisfied clients by one class (line 14) and test whether we can admit the tagged client i again. In the following, we present some important properties of the MPA admission control algorithm, including the computational complexity, and the property that it guarantees an admitted client the minimum class level that can satisfy its QoS requirement.

Lemma 3.1 The MPA admission control has a computational complexity of $O(NM^2)$.

Proof: For the MPA, we have to test whether we can admit each of the M clients (line 3). When we test the k-th client, the maximum number of clients in Ω' is equal to k (line 4). Each of these clients in Ω' may go through class upgrade (line 14) but never class downgrade. Since there are N classes in the system, we have to test $O(kN)$ configurations in the worst case. To test for all M clients, we need to test at most $\sum_{k=1}^{M} kN$ configurations, which is $O(NM^2)$.

■

In order to present the properties of MPA admission control, we first need to define the following notation and then state some preliminary results. Let $\mathbf{\Lambda} = [\Lambda_{c_1}^{max}, \cdots, \Lambda_{c_N}^{max}]$ be the arrival rate vector of different classes of requests. We define $\boldsymbol{e}_i$ as a row vector of zero with the i-th entry being one. If a client m changes its requests from class i to class j, where $j > i$, then the arrival rate vector is $\mathbf{\Lambda}' = \mathbf{\Lambda} - \lambda_m^{max}\boldsymbol{e}_i + \lambda_m^{max}\boldsymbol{e}_j$. Let $W_{c_k}(\mathbf{\Lambda})$ be the average waiting time of class k requests under loading $\mathbf{\Lambda}$.

Lemma 3.2 If a client m performs a class upgrade from class i to j $(j > i)$, then $W_{c_k}(\mathbf{\Lambda}') \geq W_{c_k}(\mathbf{\Lambda})$ for all classes $k = 1, 2, \ldots, N$.

Proof: Equation (2.6) expresses the average waiting time for each class of traffic under a PDDS system. Since $\sigma_1 = 1$ and $\sigma_i = \sigma_{i-1}/r_{i-1,i}$, we have $1 = \sigma_1 > \sigma_2 > \cdots > \sigma_N$, and so $\sigma_i > \sigma_j$. When a client m upgrades from class i to class j, it is easy to observe that the denominator of Equation (2.6) will not increase while the numerator will remain unchanged. Therefore, $W_{c_k}(\boldsymbol{\lambda}') \geq W_{c_k}(\boldsymbol{\lambda})$. ∎

Lemma 3.3 If a client m performs a class downgrade from class j to i $(i < j)$, then the average waiting times for *all* classes will not increase.

Proof: Equation (2.6) expresses the average waiting time for each class of traffic under a PDDS system. Since $\sigma_1 = 1$ and $\sigma_i = \sigma_{i-1}/r_{i-1,i}$, we have $1 = \sigma_1 > \sigma_2 > \cdots > \sigma_N$, and so $\sigma_j > \sigma_i$. When a client m downgrades from class j to class i, we can easily observe that the denominator of Equation (2.6) will not decrease while the numerator will remain unchanged. Therefore $W_{c_k}(\boldsymbol{\lambda}') \leq W_{c_k}(\boldsymbol{\lambda})$. ∎

Definition 3.4 Let $\boldsymbol{C}$ and $\boldsymbol{C}'$ be two class vectors. We say that $\boldsymbol{C} > \boldsymbol{C}'$ iff $C_i \geq C_i'$ and $\exists\ j$, where $C_j > C_j'$.

Definition 3.5 A class vector $\boldsymbol{C}$ is a feasible admitted class vector if the class assignment in $\boldsymbol{C}$ can guarantee the maximum average waiting time requirements for all admitted clients.

Definition 3.6 A minimum feasible admitted class vector $\boldsymbol{C}^*$ is a class vector such that there is no other feasible admitted class vector $\boldsymbol{C}'$ where $\boldsymbol{C}' < \boldsymbol{C}^*$.

Theorem 3.7 The MPA admission control guarantees that, at the end of every stage of testing whether to admit a client, the class vector is always a minimum feasible admitted class vector.

Proof: Let $\boldsymbol{C}^a(k)$ and $\Omega(k)$ be the admitted class vector and the set of admitted clients after testing whether we can admit the k-th client. Initially, we have $\boldsymbol{C}^a(0) = [0, \cdots, 0]$ where 0 indicates rejection and $\Omega(0) = \emptyset$. We proceed to prove the theorem by induction. Consider the first client (or $k = 1$). If MPA rejects this client because the system cannot satisfy its QoS requirement, $\boldsymbol{C}^a(1) = [0, \ldots, 0]$, which is a minimum class vector. If the system admits this client, then because MPA assigns class 1 to the client initially (line 5) and upgrades the class one at a time, the resulting admitted client vector $\boldsymbol{C}^a(1)$ is obviously a minimum feasible class vector.

Assume this property holds for $k = i - 1$. When the system tries to admit the i-th client, there are three cases to consider:

1. *The system can admit client i without changing the class assignment in $\Omega(i-1)$:* In this case, since $\boldsymbol{C}^a(i-1)$ is a minimum feasible class vector and we assign the minimum class to client i (line 5), $\boldsymbol{C}^a(i)$ is the minimum admitted class vector for $\Omega(i)$.

2. *The system cannot admit client i because there is at least one client in $\Omega(i-1)$ whose class is in class N (line 11):* In this case, client i will be rejected and we restore the previous admitted class vector (line 12). Therefore, $\boldsymbol{C}^a(i) = \boldsymbol{C}^a(i-1)$ which is the minimum admitted class vector

for $\Omega(i)$. Note that in the later stage, we may admit client j where $j > i$. But admitting client j will not make client i admissible. The reasons are (1) admitting client j will not decrease the waiting time of clients in $\Omega(i-1)$, and (2) admitting client j may result in class upgrade for clients in $\Omega(i-1)$. By Lemma 3.2, this will increase the waiting time for all clients in $\Omega(i-1)$. Since admitting client j will not decrease the waiting time of clients in $\Omega(i-1)$, if client i was not admissible, it will not become admissible after the system has admitted client j.

3. *There are $L \geq 1$ unsatisfied clients in $\Omega(i-1)$ but none of them is in class N (line 13):* In this case, the MPA will *simultaneously* upgrade the class of all these L unsatisfied clients by one (line 14). Note that we do not need to upgrade the class of each unsatisfied client sequentially. The reason is that if we upgrade one of these clients to a higher class, we increase the waiting time of all clients according to Lemma 3.2. Therefore, if we cannot satisfy the QoS of the L clients initially, a class upgrade of a subset of these clients will not make the remaining clients satisfiable. Therefore, we can simply perform a class upgrade of the L clients simultaneously. After the class upgrade, some of these clients may still be unsatisfied, in which case we repeat the process until we reach case 1 or case 2 above. Since we perform class upgrade incrementally, the resulting admitted class vector $\boldsymbol{C}^a(i)$ is a minimum feasible class vector. ∎

Remark: The implication of Lemma 3.1 and Theorem 3.7 is that not only we have a computationally efficient admission control algorithm, but the resulting admitted vector $\boldsymbol{C}^a$ is also a minimum feasible class vector. Therefore, we can ensure that we can provide QoS guarantees to all admitted clients and, at the same time, not overcharge these clients by assigning them to higher classes than needed.

The MPA algorithm assumes that a client with a tighter QoS requirement (i.e., smaller maximum average waiting time) is more willing to pay a higher cost for the web service. On the other hand, a web server operator may want to maximize the number of admitted clients so as to popularize the web service. In this case, we propose the following admission control algorithm.

3.3 Maximum Admission Algorithm (MAA)

The second admission control algorithm is the MAA. The objective is to admit as many clients as possible into the web server. The rationale is that by admitting more clients, the web service will be more popular and the content provider will be able to charge more and generate more profit in the long run. Under MAA, we try to admit those clients with a less stringent QoS requirement (i.e., large maximum average waiting time) first. The MAA algorithm is given as:

MAA Admission Control

1. Sort the maximum average waiting time requirement of all clients from largest to smallest. If there is a tie, sort clients based on the maximum arrival rate from smallest to largest. Assume that client 1 (respectively, client M) has the largest (respectively, smallest) maximum average waiting time requirement.
2. Let Ω be the set of all admitted clients. Initialize $\Omega = \emptyset$ and the admitted class vector $\boldsymbol{C}^a = \mathbf{0}$;
3. **for** ($i = 1$ **to** M) {`/* test all` M `clients */`
4. $\Omega' = \Omega \bigcup$ client i; $\boldsymbol{C}^a_{temp} = \boldsymbol{C}^a$; satisfied_flag = false;
5. assign client i to class 1;
6. **while** (satisfied_flag == false) {
7. compute delay of all clients in Ω' based on Eq.(2.6);

8. **if** (waiting times of all clients in Ω' are satisfied) {
9. $\Omega = \Omega'$; satisfied_flag=true; }
10. **else**{/* perform class upgrade for unsatisfied clients*/
11. **if** (there is any unsatisfied client with class equal N) {
/*can't admit client i, restore $\boldsymbol{C}^a$*/
12. $\boldsymbol{C}^a = \boldsymbol{C}^a_{temp}$; satisfied_flag = true;
min_arrival_rate=arrival rate of client i;
$i^* = i; i = M;$}
13. **else** /* upgrade unsatisfied clients */
14. increase the class of all unsatisfied client in Ω' by 1;
15. }
16. } /* termination of while loop */
17. } /* termination of for loop */
/* test whether we can admit client $i^* + 1$ to M */
18. **for** ($i = i^*$+1 **to** M) {
19. **if** (arrival rate of client $i <$ min_arrival_rate) {
20. $\Omega' = \Omega \bigcup$ client i; $\boldsymbol{C}^a_{temp} = \boldsymbol{C}^a$; satisfied_flag = false;
21. assign client i to class 1;
22. **while** (satisfied_flag == false) {
23. compute delay of all clients in Ω' based on Eq.(2.6);
24. **if** (waiting times of all clients in Ω' are satisfied) {
25. $\Omega = \Omega'$; satisfied_flag=true; }
26. **else**{/* perform class upgrade for unsatisfied clients*/
27. **if** (there is any unsatisfied client with class equal N) {
/*can't admit client i, restore $\boldsymbol{C}^a$*/
28. $\boldsymbol{C}^a = \boldsymbol{C}^a_{temp}$; satisfied_flag = true;
min_arrival_rate = arrival rate of client i;}
29. **else** /* upgrade unsatisfied clients */
30. increase the class of all unsatisfied client in Ω' by 1;

```
31.       }
32.     } /* termination of while loop */
33.   } /* termination for if loop */
34. } /* termination of for loop */
35. return (Ω and C^a);
```

Let us explain the rationale of the MAA algorithm. Under MAA admission control, we test whether we can admit a tagged client (line 3). For this tagged client i, we first assign it to class one (line 5). By adding this tagged client i, we may change the waiting times of previously admitted clients. We test whether this new additional client will violate the QoS of other clients in Ω' (line 7). If the addition does not violate the QoS of any client, we can admit this tagged client i (line 9). On the other hand, if there is any QoS violation and the unsatisfied clients are already in class N, this implies that we cannot admit the tagged client i (line 11-12). If there is QoS violation and none of the unsatisfied clients is in class N, we can upgrade all these unsatisfied clients by one class (line 14) and test whether we can admit the tagged client i again. Once we find the first client that we cannot admit (we call this client i^*), we go to the second phase of the algorithm by testing whether we can admit the remaining clients (clients $i^* + 1$ to M). Because of the initial sorting, the remaining clients will have a maximum average waiting time requirement smaller than or equal to that of client i^*. Therefore, we can do much pruning by skipping those clients whose arrival rates are larger than the arrival rate of client i^* because the server $\mathcal{S}$ cannot admit these clients for sure. In the following, we present some important properties of the MAA admission control: its computational complexity and the property of guaranteeing an admitted client the minimum service class for its QoS requirement.

Lemma 3.8 *MAA admission control has a computational complexity of* $O(NM^2)$.

Proof: Under the MAA, we have to test whether we can admit each of the M clients (lines 3). The algorithm is divided into two phases, (a) lines $3-17$ and (b) lines $18-34$. In the first phase, when we test whether we can admit the k-th client, the number of clients in Ω' is equal to k. Each of these clients in Ω' may go through class upgrade (line 14) but never class downgrade. Since there are N classes in the system, we have to test $O(kN)$ configurations in the worst case. The end of the first phase is signaled by an unsatisfied client in class N (line 11-12). There are at most M clients to be tested in the first phase. We need to test at most $\sum_{k=1}^{M} kN$ configurations, which is $O(NM^2)$. When the unsatisfied client (we call this client i^*) is found in the first phase, its arrival rate is marked so that we can perform pruning in the second phase. In the second phase, a user will be tested only if its arrival rate is less than the arrival rate of client i^*. We can perform this pruning because for client $j > i^*$, we have $W_j^{max} < W_{i^*}^{max}$ (due to initial sorting) and if $\lambda_j^{max} > \lambda_{i^*}^{max}$, the server $\mathcal{S}$ cannot admit client j for sure. If client j has a smaller arrival rate than client i^*, the maximum number of clients in Ω' is equal to j, and each of these clients in Ω' may go through class upgrade (line 30) but never class downgrade. Since there are N classes in the system, we have to test $O(jN)$ configurations in the worst case. As there are at most M clients to be tested in second phase, we need to test at most $\sum_{k=1}^{M} kN$ configurations, which is $O(NM^2)$. Therefore, the worst case computational complexity is $O(NM^2)$ for the MAA algorithm. ∎

Theorem 3.9 MAA admission control guarantees that, at the end of every stage of testing whether to admit a client, the class vector is always a minimum feasible admitted class vector.

Proof: Let $\boldsymbol{C}^a(k)$ and $\Omega(k)$ be the admitted class vector and the set of admitted clients after testing whether we can admit the k-th client. Initially, we have $\boldsymbol{C}^a(0) = [0, \cdots, 0]$ where 0 indicates rejection and $\Omega(0) = \emptyset$. We proceed to prove the theorem by induction. Consider the first client (or $k = 1$). If MAA rejects this client because the system cannot satisfy its QoS requirement, $\boldsymbol{C}^a(1) = [0, \ldots, 0]$, which is a minimum class vector. If the system admits this client, then because MAA assigns class 1 to this client initially (line 5) and upgrades its class by one at a time, the resulting admitted client vector $\boldsymbol{C}^a(1)$ is obviously a minimum feasible class vector.

Assume the property holds for $k = i-1$. When the system tries to admit the i-th client, we divide the process into two phases: The first phase corresponds to lines $3 - 17$ while the second phase corresponds to lines $18 - 34$.

In the first phase, there are three cases to consider:

1. *The system can admit client i without changing the class assignment in $\Omega(i-1)$:* In this case, since $\boldsymbol{C}^a(i-1)$ is a minimum feasible class vector and we assign the minimum class to client i (line 5), $\boldsymbol{C}^a(i)$ is the minimum admitted class vector for $\Omega(i)$.

2. *The system cannot admit client i because there is at least one client in $\Omega(i-1)$ whose class is in class N (line 11):* In this case, client i will be rejected and we restore the previous admitted class vector (line 12). Therefore, $\boldsymbol{C}^a(i) = \boldsymbol{C}^a(i-1)$ which is the minimum admitted class vector for $\Omega(i)$. Note that in the later stage, we may admit client j, where $j > i$. But admitting client j will not make client i to be admissible. The reasons are (1) admitting client j will not decrease the waiting time of clients in $\Omega(i-1)$, and (2) admitting client j may result in class upgrade for the clients in $\Omega(i-1)$. By Lemma 3.2, this will increase the waiting time for all clients in $\Omega(i-1)$. Since admitting client j will not decrease the waiting times of clients in $\Omega(i-1)$, if client i was not admissible, it

will not become admissible after the system has admitted client j.

3. *There are $L \geq 1$ unsatisfied clients in $\Omega(i-1)$ but none of them is in class N (line 13):* In this case, the MAA will *simultaneously* upgrade the class of all these L unsatisfied clients by one (line 14). Note that we do not need to upgrade the class of each unsatisfied client sequentially. The reason is that if we upgrade one of these clients to a higher class, we increase the waiting time of all clients according to Lemma 3.2. Therefore, if we cannot satisfy the QoS of the L clients initially, a class upgrade of a subset of these clients will not make the remaining clients to be satisfiable. Therefore, we can simply perform a class upgrade of the L clients simultaneously. After the class upgrade, some of these clients may still be unsatisfied, in which case we repeat the process until we reach case 1 or case 2 above. Since we perform class upgrade incrementally, the resulting admitted class vector $\boldsymbol{C}^a(i)$ is a minimum feasible class vector.

Hence, the resulting admitted client vector is a minimum feasible class vector after the first phase. In the second phase, we have to consider two cases:

1. $\lambda_i^{max} \geq$ *minimum_arrival_rate:* In this case, there must be a last-rejected client, say k. Because of the sorting on the maximum average waiting time requirement, we have $W_i^{max} \leq W_k^{max}$. Noting that clients can only do class upgrade but not downgrade, we only need to prove a class vector is infeasible for $\Omega(k-1) \bigcup i$ if it is infeasible for $\Omega(k)$.

   Since $\lambda_i^{max} \geq$ minimum_arrival_rate, by equation (2.6), the average waiting time of all classes must be increased or remain unchanged. As it is not a feasible class vector for client k, there must be at least one client that is not satisfied with its waiting time requirement. If that unsatisfied client is client k, then client i must also be unsatisfied due to $W_i^{max} \leq W_k^{max}$. If that unsatisfied client is another client l (where $l \neq k$), then client

l is also unsatisfied since its requirement remains unchanged while the waiting time increases.

2. $\lambda_i^{max} < minimum\_arrival\_rate$: In this case, the MAA will test whether client i can be accepted by the system. As the procedure is just as in the first phase, the proof of correctness is similar to the one for the first phase.

Hence, the resulting admitted client vector is a minimum feasible class vector after the second phase. We will present the performance of these two admission control algorithms in Chapter 5. ∎

Chapter 4

Dynamic Class Adaptation

Based on the admission control algorithms proposed in Chapter 3, the PDDS-enabled web server $\mathcal{S}$ can provide QoS guarantees to all the admitted clients. In other words, the expected waiting time of each client is guaranteed to be upper bounded by its specified maximum average waiting time. One important point to observe is that the admission control is carried out based on the *maximum* arrival rate specified by each client. It is possible that the average arrival rate of the admitted client is less than or equal to its specified maximum arrival rate. Let λ_j denote the average arrival rate of the admitted client j. If

$$\sum_{j=1}^{M'} \lambda_j < \sum_{j=1}^{M'} \lambda_j^{max},$$

it implies that there is an opportunity for an admitted client, say j, to submit requests to the PDDS-enabled web server $\mathcal{S}$ with a class value which is less than C_j^a and still able to attain its QoS requirement (i.e., the average waiting time is less than W_j^{max}). In this chapter, we propose two dynamic adaptation algorithms so that the admitted clients can dynamically adapt to the system loading at $\mathcal{S}$.

Before we present these two dynamic adaptation algorithms, let us present the general framework wherein the PDDS-enabled web server $\mathcal{S}$ can measure the necessary information and send feedback control information back to all admitted clients. Figure 4.1 illustrates the general framework. Assume that

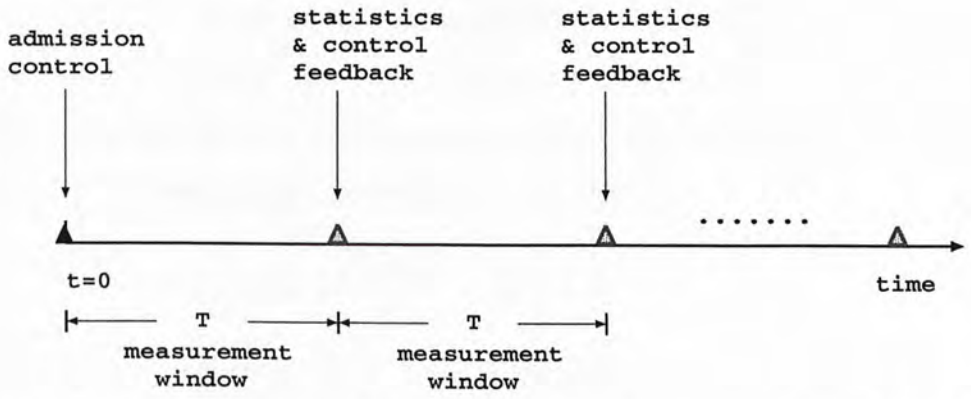


Figure 4.1: General framework for server $\mathcal{S}$ to collect statistics and send feedback control back to clients.

the server $\mathcal{S}$ has completed the admission control process (either via MPA or MAA) at time $t = 0$. Each admitted client will submit requests to $\mathcal{S}$ based on its class assignment in $\boldsymbol{C}^a$. For every measurement window of length T, the server $\mathcal{S}$ measures the request arrival rates. At the end of each period, the server $\mathcal{S}$ either sends back a new class vector $\boldsymbol{C}$ to all the admitted clients, or sends back the arrival statistics to all the admitted clients, who can then perform their own class adaptation.

4.1 Centralized Approach: Server-Based Dynamic Adaptation (SBDA)

Under server-based adaptation, the web server estimates the arrival rate of each client within a measurement window, and then computes a new class vector for each admitted client at the end of the measurement period. The new class assignment will be sent to each admitted client. Each admitted client can then submit requests to the PDDS-enabled web server in a class range that is between the new class value and the original admitted class value.

Formally, let $\boldsymbol{C}(nT)$ denote the class vector at the end of the n-th measurement period. We have $\boldsymbol{C}(0) = \boldsymbol{C}^a$, the initial class vector after the admission

control process. Within a measurement window, the server $\mathcal{S}$ estimates the arrival rate of client j. Let $N_j(nT)$ be the number of requests submitted by client j during the n-th measurement period. The estimated arrival rate of client j at the end of this measurement period is:

$$\hat{\lambda}_j = \frac{N_j(nT)}{T} \qquad j = 1, 2, \ldots, M'. \tag{4.1}$$

To generate a new class vector $\boldsymbol{C}(nT)$, the server can use either the MPA or the MAA algorithm described in the previous chapter. Once the new class vector is computed, the server $\mathcal{S}$ sends the new class value $C_j(nT)$ to client j, $j = 1, 2, \ldots, M'$.

Upon receiving the new class value, the client j can choose to tag the request in class C_j^* where $C_j(nT) \leq C_j^* \leq C_j^a$. Here, we consider that a client j will initially tag its requests as $C_j(nT)$. During the process of request submission, client j also estimates its waiting time. If it is more than the maximum average waiting time requirement W_j^{max}, then client j will upgrade its requests by one class. The maximum class value that class j can tag its requests is C_j^a. Note that if the estimated average waiting time is less than W_j^{max}, then client j will not perform any class downgrade and will continue to submit requests based on the current class value. This way, client j can reduce its usage cost for $\mathcal{S}$. Figure 4.2 illustrates an example in which client j performs a class upgrade at instants τ_1, τ_2, τ_3, and τ_4.

We like to stress two important points here:

- First, $C_j(nT)$ is guaranteed to be less than or equal to C_j^a. The reason is that the original class vector $\boldsymbol{C}(0)$ (or $\boldsymbol{C}^a$) was computed based on the "maximum" arrival rate of each admitted client. Since the arrival rates of all admitted clients within the measurement period are less than or equal to their maximum arrival rates, the resulting class vector $\boldsymbol{C}(nT)$ is guaranteed to be less than or equal to $\boldsymbol{C}^a$.

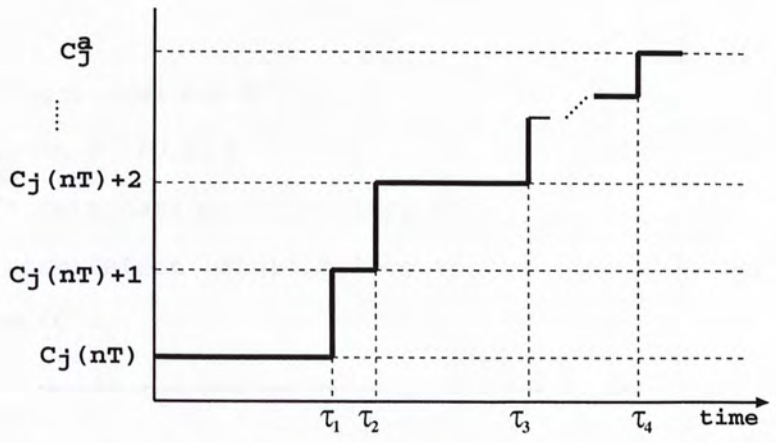


Figure 4.2: Class adaptation by client j within a measurement period.

- Secondly, if client j tags its requests in class C_j^a, j is assured that its requests will definitely satisfy its QoS requirement.

The procedure of SBDA is given as:

SBDA Algorithm

1. **for** ($i = 1$ to M') {/* test all M' clients */
2. $\quad \boldsymbol{C}_i' = \boldsymbol{C}_i^a$;
3. } /* termination of for loop */
4. finish = FALSE;
   num_series = 0;
   compute delay of all classes using C';
5. **while** (finish == FALSE) {
6. $\quad$ **for** ($i = 1$ to M') {/* test all M' clients */
7. $\quad\quad$ num_series++;
8. $\quad\quad$ **if** ($\boldsymbol{C}_i' > 1$) {
9. $\quad\quad\quad$ **if** ($W_{C_i'-1} < W_i^{max}$) {
10. $\quad\quad\quad\quad$ $\boldsymbol{C}_i' = \boldsymbol{C}_i' - 1$;
    $\quad\quad\quad\quad$ num_series = 1;
    $\quad\quad\quad\quad$ compute delays of all classes using C'; }

```
11.      }
12.      if (num_series == M') {
13.        finish = TRUE; }
14.    } /* termination of for loop */
15. } /* termination of while loop */
16. return (C');
```

If the future arrival rates of all admitted clients will not change, the SBDA can find the minimum feasible admitted class vector. The SBDA assigns each client i to C_i^a first. Every client then tries to do class downgrade sequentially. If a client finds the average waiting time of class C'_{i-1} where $i - 1 \geq 1$ can still satisfy its maximum average waiting time requirement, the client will perform class downgrade. When all admitted clients stop downgrading, the process terminates. The class vector C' computed satisfies the waiting time requirements of all admitted clients, and allow them to submit at the lowest possible class. In the following, we present some important properties of the SBDA.

Theorem 4.1 If the future arrival rates of all admitted clients will not change, the class vector computed in SBDA satisfies the waiting time requirements of all admitted clients, and these clients will submit at the lowest possible class.

Proof: When a client tests if it needs to perform a class downgrade (line 8-9), there are only 2 possibilities. Either (1) it moves to one class lower if it finds that its QoS requirement can be satisfied in the lower class, or (2) it remains in the same class if the requirement cannot be satisfied in the lower class. In case (1), when a client performs a class downgrade, the average waiting time of other classes will not increase by Lemma 3.3. Hence, no user needs to perform class upgrade. In case (2), when a client remains in the same class, this

causes no change to the average waiting times of other classes and so no client needs to do class upgrade. Therefore, the waiting time requirements of all the admitted clients are guaranteed. ∎

Lemma 4.2 *The SBDA has a computational complexity of* $O(NM'^2)$.

Proof: In worst case, for a loop of M' clients, there is only one client needs to perform a class downgrade. As there are M' clients and N classes, the maximum number of class downgrade is NM'. Therefore, the worst case computation complexity is $O(NM'^2)$. ∎

There are some major drawbacks about the server-based dynamic adaptation approach. For example, it is computationally expensive to estimate the arrival rate of *each* admitted client in Equation (4.1). Another disadvantage is that the server $\mathcal{S}$ needs to send the new class value $\boldsymbol{C}(nT)$ to each admitted client, which implies that the server needs to perform M' operations to reach all the admitted clients. On the other hand, the advantage of the SBDA approach is that the new class vector $\boldsymbol{C}(nT)$ is very precise. If there is no major change in the future workload, then each admitted client will pay the lowest usage cost and still be able to receive service within its QoS requirement.

4.2 Distributed & Game-Theoretic Approach: Client-Based Dynamic Adaptation (CBDA)

The SBDA algorithm can be computationally expensive, both in tracking the arrival rates of all M' clients and in sending the new class vector to all the clients. We propose an alternative client-based dynamic adaptation algorithm (CBDA), which is a *distributed adaptation* algorithm wherein each client can choose the appropriate class in submitting its requests.

Unlike the SBDA algorithm, the web server $\mathcal{S}$ does not need to track the arrival rate of each admitted client, but rather, estimate the arrival rates of individual classes of requests. Therefore, rather than track M' variables as in SBDA, CBDA only tracks N variables. Since $N << M'$, this results in a major saving for the computation overhead. Let $\tau_{c_k,n}$ be the interarrival time between the $(n-1)$-th and the n-th requests in class k. We use an exponential weighted time average method to estimate $\hat{\Lambda}_{c_k}(n)$, the arrival rate of class k at the n-th request arrival. The estimation is

$$\hat{\Lambda}_{c_k}(n) = (1-\sigma)\hat{\Lambda}_{c_k}(n-1) + \sigma(\tau_{c_k,n})^{-1} \quad k = 1, ..., N \tag{4.2}$$

where $0 \leq \sigma \leq 1$. At the end of each measurement period, the web server $\mathcal{S}$ *multicasts* this class vector $\hat{\mathbf{\Lambda}}_c = [\hat{\Lambda}_{c_1}, \hat{\Lambda}_{c_2}, \cdots, \hat{\Lambda}_{c_N}]$ to all the M' admitted clients.

Each client, upon receiving the new class vector $\hat{\mathbf{\Lambda}}_c$, can determine the minimum class for its future requests. To illustrate, consider that client j receives $\hat{\mathbf{\Lambda}}_c$ from the server $\mathcal{S}$. Let $\hat{\lambda}_{j,c_k}$ be the class k traffic rate submitted by client j in the previous measurement period. Then, the traffic rate vector of client j in the previous measurement period is $\hat{\boldsymbol{\lambda}}_j = [\hat{\lambda}_{j,c_1}, \hat{\lambda}_{j,c_2}, \cdots, \hat{\lambda}_{j,c_N}]$. Upon receiving $\hat{\mathbf{\Lambda}}_c$, client j executes the following code:

Adaptation algorithm for client j

```
/*compute rate from previous round*/
```

1. Let $\tilde{\lambda}_j = \sum_{k=1}^{N} \hat{\lambda}_{j,c_k}$;
2. **for** ($k = 1$ **to** C_j^a) {
3. `/*try new rate vector` $\mathbf{\Lambda}^*$ `in class` k `*/`
4. $\quad \mathbf{\Lambda}^* = \hat{\mathbf{\Lambda}}_c - \hat{\boldsymbol{\lambda}}_j + \tilde{\lambda}_j \boldsymbol{e}_k$;
5. $\quad$ Based on Eq.(2.6) and $\mathbf{\Lambda}^*$, compute delay for client j;
6. $\quad$ **if** (computed delay $\leq W_j^{max}$)
7. $\quad\quad$ selected_class $= k$; $k = C_j^a$;

8. } /* terminate for loop */
9. **return** (selected_class);

Lemma 4.3 *The CBDA has a computational complexity of $O(N)$ for each client.*

Proof: Since an admitted client j only needs to test from class 1 to class C_j^a, and C_j^a is upper bounded by N, so the worst case computation complexity for an admitted client is $O(N)$. ∎

In other words, client j tries to maximize its utility by finding the lowest class such that the average waiting time is less than or equal to the maximum average waiting time requirement, W_j^{max}. In essence, this is a *non-cooperative game* problem in which distributed optimization is performed by each client. For an introduction to the basic concepts of game theory, please refer to [6]. We assume that clients ignore how they influence the class adaptation of other clients when optimizing their own utility. This simplifying assumption corresponds to the standard competitive price taking assumption of economic theory. Also, the above assumption can be justified when

1. The traffic loading of an individual client is considered to be *small*, as compared to the overall traffic loading at the web server, so that the class adaptation by a client is considered to be negligible.

2. It is impractical or computationally expensive for a client to determine how to perform class adaptation based on all the other clients' class adaptation decisions.

There are several important properties of the CBDA algorithm, as follows:

- **Guaranteed termination:** Each client j searches for the lowest suitable class, from class 1 to C_j^a. In the worst case, the algorithm will terminate when the class is equal to C_j^a, which is the assigned class during the admission control process. The reason is that the admission control decision was made based on the specified maximum arrival rates for all clients. Therefore, if client j is admitted, by selecting its class equal to C_j^a, we can guarantee that the QoS requirement of client j will be met.

- **Low computational complexity:** Unlike the SBDA approach where the server has to track the arrival rates of *all* M' clients and then *recompute* a new class vector (in essence, re-execute the admission control algorithm), the workload under the CBDA approach is *distributed* among all the clients. The server $\mathcal{S}$ only needs to track the arrival rates for N classes and class adaptation is carried out by the individual clients. If some clients do not want to perform class adaptation, they can simply ignore this optimization step.

Of course, one can argue that the adaptation based on the SBDA algorithm is more *precise* than the CBDA algorithm because it uses all available information (i.e., arrival rates of all clients) in making an adaptation decision. We illustrate the performance difference between the two algorithms in the next chapter.

Chapter 5

Performance Evaluation

In this chapter, we compare the performance of the MPA and MAA admission control algorithms. We also present performance results for the SBDA and CBDA adaptation algorithms under various settings. For example, different arrival rates under Poisson, MMPP and Pareto arrival process, different waiting time spacing ratios (i.e., $r_{i,i+1}$), different utility functions, ... etc.

5.1 Experiment 1: Comparison of MPA and MAA Admission Control

In this experiment, we compare the performance of the MPA and MAA algorithms. In particular, the performance metrics that we are interested are (1) the number of admitted clients M', (2) the admitted arrival rates of different classes, (3) the achieved waiting times for different classes of requests, and (4) the achieved system efficacy. Unless otherwise stated, we assume that the service times of all the requests are exponentially distributed with mean equal to unity. The aggregate request rate from all clients is modeled as a Poisson process with rate λ_r. Note that λ_r is the workload *before* the admission control procedure. The PDDS-enabled web server supports $N = 3$ classes of requests and their waiting time differentiations are $r_{1,2} = 1.4, r_{2,3} = 1.4$.

# of clients	MPA	MAA
$M = 1,000$	$M' = 497$	$M' = 832$
$M = 2,000$	$M' = 993$	$M' = 1{,}663$
$M = 5,000$	$M' = 2{,}479$	$M' = 4{,}155$

Table 5.1: MPA vs. MAA for the number of admitted clients M'.

5.1.1 Experiment 1.A

We vary the number of potential clients that want to access the server $\mathcal{S}$ to be M=1000, 2000, and 5000. The maximum average waiting time requirements of these clients are drawn uniformly between $[1.5, 5.5]$ seconds and the aggregate request rate λ_r is set to one. Since this arrival rate can saturate the system ($\rho = 1$), it is necessary for us to perform admission control. Table 5.1 illustrates the total number of admitted clients M' for the MPA and MAA algorithms under different values of M. We can see that MAA can admit more clients because this algorithm tries to admit clients with less stringent maximum average waiting time requirements first. This also indicates that, if the admission cost is fixed on a per class basis, it makes sense to use the MAA algorithm so as to maximize the total admission revenue.

	class #	MPA	MAA
$\lambda_r = 0.5$	class 3	—	—
	class 2	—	—
	class 1	0.500	0.500
$\lambda_r = 0.75$	class 3	0.058	0.058
	class 2	0.138	0.138
	class 1	0.554	0.554
$\lambda_r = 1.0$	class 3	0.145	0.188
	class 2	0.189	0.233
	class 1	0.432	0.402

Table 5.2: MPA vs. MAA: arrival rates of different classes. Note that for $\lambda_v = 0.5$, all clients are assigned to class 1.

	class #	MPA	MAA
λ_r = 0.5	class 3	—	—
	class 2	—	—
	class 1	1.008 (2.000,9.991)	1.008 (2.000,9.991)
	$r_{1,2}$	—	—
	$r_{2,3}$	—	—
λ_r = 0.75	class 3	1.701 (2.000,2.378)	1.701 (2.000,2.378)
	class 2	2.381 (2.387,3.331)	2.381 (2.387,3.331)
	class 1	3.334 (3.336,9.991)	3.334 (3.336,9.991)
	$r_{1,2}$	$r_{1,2} = 1.4$	$r_{1,2} = 1.4$
	$r_{2,3}$	$r_{2,3} = 1.4$	$r_{2,3} = 1.4$
λ_r = 1.0	class 3	2.000 (2.000,2.788)	2.950 (2.982,4.119)
	class 2	2.800 (2.802,3.916)	4.130 (4.133,5.776)
	class 1	3.920 (3.926,7.192)	5.782 (5.790,9.991)
	$r_{1,2}$	$r_{1,2} = 1.4$	$r_{1,2} = 1.4$
	$r_{2,3}$	$r_{2,3} = 1.4$	$r_{2,3} = 1.4$

Table 5.3: MPA vs. MAA: achieved average waiting times of different classes. The numbers in parenthesis indicate the two extremes (i.e., the most stringent and the least stringent) of the maximum waiting time requirements of the admitted clients in that particular class.

5.1.2 Experiment 1.B

We set the number of potential clients M to 1000. We vary the aggregate request arrival rate λ_r as 0.5, 0.75 and 1.0. The maximum average waiting time requirements of all clients are drawn uniformly between $[2, 10]$ seconds. Table 5.2 and Table 5.3 illustrate that, after the admission control and client classification, the arrival rates and the achieved waiting times of the 3 classes of requests. From Table 5.2, we observe that at low and moderate workload (e.g., $\lambda_r = 0.5$ or 0.75), both the MPA and MAA can effectively assign clients to the appropriate class so that these admitted clients will pay the lowest possible

usage cost. For example, at low workload ($\lambda_r = 0.5$), both algorithms assign all clients to class 1 (therefore, it becomes single queue scheduling). Under single queue scheduling, the achieved waiting time will be less than the maximum waiting time requirements of all clients. When the system is under high workload ($\lambda_r = 1$), MPA and MAA can filter out those clients whose maximum average waiting time requirements are unrealizable and classify admitted clients to the lowest admissible class.

Table 5.3 depicts the achieved waiting times for different classes under the MPA and MAA algorithms. The numbers in parenthesis indicate the two extremes (i.e., the most stringent and the least stringent) of the maximum average waiting time requirements of the admitted clients in that particular class. For example, under $\lambda_r = 0.5$ and MPA, the most stringent (least stringent) maximum average waiting time requirement is 2.000 (9.991) seconds and the achieved waiting time is 1.008 seconds. From Table 5.3, we observe that both the MPA and MAA algorithms can effectively classify clients to the lowest admissible classes so that their QoS can be satisfied. At the same time, the achieved waiting time ratio is equal to the specified ratio of $r_{i,i+1} = 1.4$.

5.1.3 Experiment 1.C

We compare the effectiveness of the MPA and MAA algorithms under different waiting time spacings. We vary the waiting time spacing $r_{i,i+1}$ to be 1.3, 1.4 and 1.5. The aggregate request arrival rate is $\lambda_r = 1.0$ and the maximum average waiting time requirements of the clients are drawn uniformly from $[2.0, 5.0]$ seconds. From Table 5.4, we observe that both the MPA and the MAA algorithms can effectively classify clients to the lowest admissible classes so that their QoS requirements are satisfied. At the same time, the achieved waiting time ratio is very close to the specified waiting time ratio $r_{i,i+1}$.

	class #	MPA	MAA
$r_{i,i+1}$ = 1.3	class 3	2.000 (2.000,2.597)	2.502 (2.512,3.245)
	class 2	2.600 (2.600,3.377)	3.253 (3.255,4.227)
	class 1	3.380 (3.382,3.772)	4.229 (4.230,4.997)
	W_1/W_2	1.300	1.300
	W_2/W_3	1.300	1.300
$r_{i,i+1}$ = 1.4	class 3	1.556 (2.000,2.174)	1.881 (2.527,2.628)
	class 2	2.178 (2.179,3.044)	2.633 (2.634,3.683)
	class 1	3.049 (3.050,3.743)	3.686 (3.687,4.997)
	W_1/W_2	1.400	1.400
	W_2/W_3	1.400	1.400
$r_{i,i+1}$ = 1.5	class 3	1.338 (2.000,2.002)	—
	class 2	2.007 (2.007,3.005)	2.484 (2.533,3.722)
	class 1	3.011 (3.011,3.733)	3.725 (3.726,4.997)
	W_1/W_2	1.500	—
	W_2/W_3	1.500	1.500

Table 5.4: MPA vs. MAA: achieved waiting times of different classes under different waiting time spacings $r_{i,i+1}$. The numbers in parenthesis indicate the two extremes (i.e., the most stringent and the least stringent) of the maximum waiting time requirements of the admitted clients in that particular class.

5.1.4 Experiment 1.D

This experiment illustrates the effect of utility function on the system efficacy. The usage cost of each class corresponds to the waiting time ratio; e.g., if $r_{1,2} = 1.4$, the usage costs of classes 1 and 2 are 1.0 and 1.4, respectively. The general form of utility function is

$$U(A, B, C, D, wt) = D \times \tan^{-1}(\frac{C - wt}{A}) + B \tag{5.1}$$

where wt is the achieved waiting time of the client, and A, B, C, D are the utility function parameters. (The detailed description of these parameters are

	utility function	**MPA**	**MAA**
$r_{1,2} = 1.3$ $r_{2,3} = 1.4$	U1	1311.930	1429.809
	U2	1344.348	1466.720
	U3	1293.506	1417.253
	U4	1098.027	1115.817
$r_{1,2} = 1.4$ $r_{2,3} = 1.5$	U1	1315.466	1432.379
	U2	1350.890	1469.881
	U3	1299.807	1420.757
	U4	1101.362	1120.585
$r_{1,2} = 1.5$ $r_{2,3} = 1.6$	U1	1313.554	1439.298
	U2	1350.200	1470.294
	U3	1299.550	1422.468
	U4	1101.542	1127.962

Table 5.5: MPA vs. MAA: system efficacy under different utility functions and waiting time spacings $r_{i,i+1}$.

presented in Appendix A). The number of potential clients M is set to 1000. The aggregate request arrival rate λ_r is 1.2. The maximum average waiting time requirements of clients are drawn uniformly between [6, 20] seconds. Table 5.5 illustrates the system efficacy of admitted clients for the MPA and MAA algorithms with different utility functions. First, we compare the effect of sensitivity of waiting time on system efficacy. Utility templates U1, U2, and U3 have the same average utility and maximum increase in utility, but U1 is twice as sensitive on waiting time as U2, while U2 is twice as sensitive as U3. We see that the system efficacy is the largest in U2 both for MPA and MAA. But MAA always give a larger system efficacy than MPA. For the comparison between utility templates U1 and U4 (both U1 and U4 have the same sensitivity and average utility, but U1 has a larger maximum increase in utility than U4), we see that the decrease of system efficacy in MAA is larger

than that in MPA. Hence, the "maximum increase in utility" parameter seems to have a larger effect on MAA than MPA.

5.1.5 Experiment 1.E

		MPA	MAA	Exhaustive Search
$r_{1,2} = 1.3$ $r_{2,3} = 1.4$	$N = 3$, $M = 13$	12.272	15.075	15.324
	$N = 3$, $M = 15$	13.681	17.048	17.708
	$N = 4$, $M = 13$	12.272	15.075	15.324
$r_{1,2} = 1.4$ $r_{2,3} = 1.5$	$N = 3$, $M = 13$	12.272	15.075	15.378
	$N = 3$, $M = 15$	13.518	17.035	17.848
	$N = 4$, $M = 13$	12.272	15.075	15.378
$r_{1,2} = 1.5$ $r_{2,3} = 1.6$	$N = 3$, $M = 13$	12.272	15.075	15.414
	$N = 3$, $M = 15$	14.018	17.195	17.905
	$N = 4$, $M = 13$	12.272	15.075	15.414

Table 5.6: MPA, MAA vs. exhaustive search: system efficacy under different number of priority classes, potential clients and waiting time spacings $r_{i,i+1}$.

We compare the effectiveness of MPA and MAA algorithms with exhaustive search under different number of priority classes, potential clients and waiting time spacings. The aggregate request arrival rate is $\lambda_r = 1.0$ and the maximum average waiting time requirements of the clients are drawn uniformly from $[6.0, 20.0]$ seconds. Table 5.6 illustrates the system efficacy of admitted clients for the MPA, MAA algorithms and exhaustive search with utility template U1. From Table 5.6, we observe that both the MPA and the MAA algorithms can effectively classify clients to obtain a system efficacy close to the optimal solution. In this experiment, MAA algorithm performs better than MPA algorithm because the utility functions of clients are drawn according to the rationale of MAA.

5.2 Experiment 2: Necessity for Adaptation

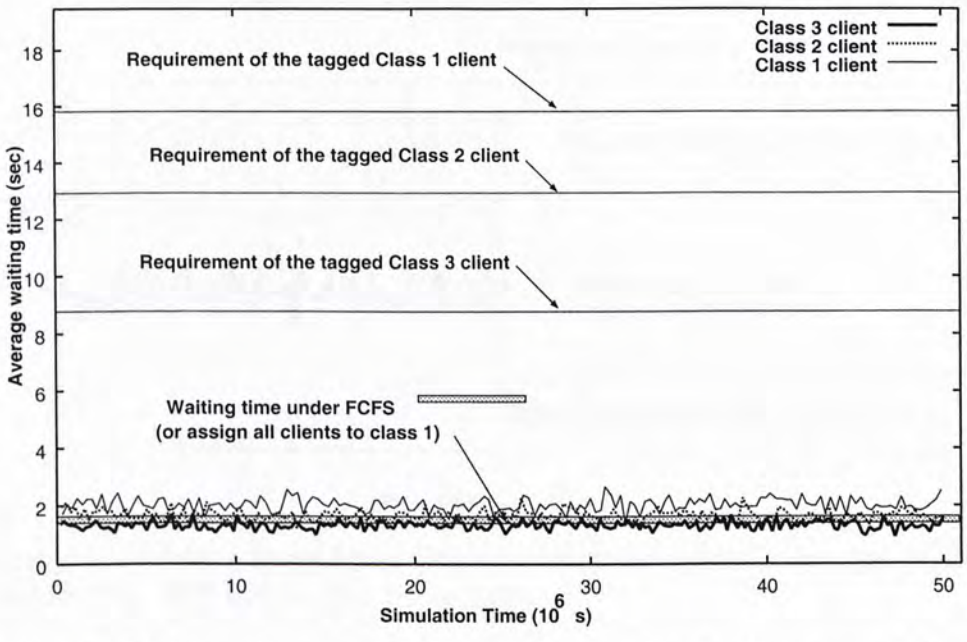


Figure 5.1: Waiting times for 3 different clients wherein the aggregated workload is only half of the maximum specification

In this experiment, we illustrate the necessity to perform class adaption. Consider the scenario wherein after the admission control at $t = 0$, all admitted clients only submit requests to the PDDS-enabled web server at rates which are only 70% of their maximum arrival rates (or λ_j^{max}). Figure 5.1 depicts the waiting times of three "tagged" clients from *each* of the three classes as well as their maximum average waiting time requirements. Each point of the plot is the average waiting times of the previous 200 requests of the tagged client. As we can observe, rather than enforce each client to submit requests based on the assigned class, we can simply serve all requests in the system based on a FCFS policy. In other words, if all clients submit requests in class 1, the system can still satisfy all their QoS requirements. This illustrates the necessity of dynamic adaptation. Because the instantaneous workload at the server $\mathcal{S}$ is less than the maximum specified workload, if there is class adaptation, different clients can pay a lower usage cost while still satisfying their QoS.

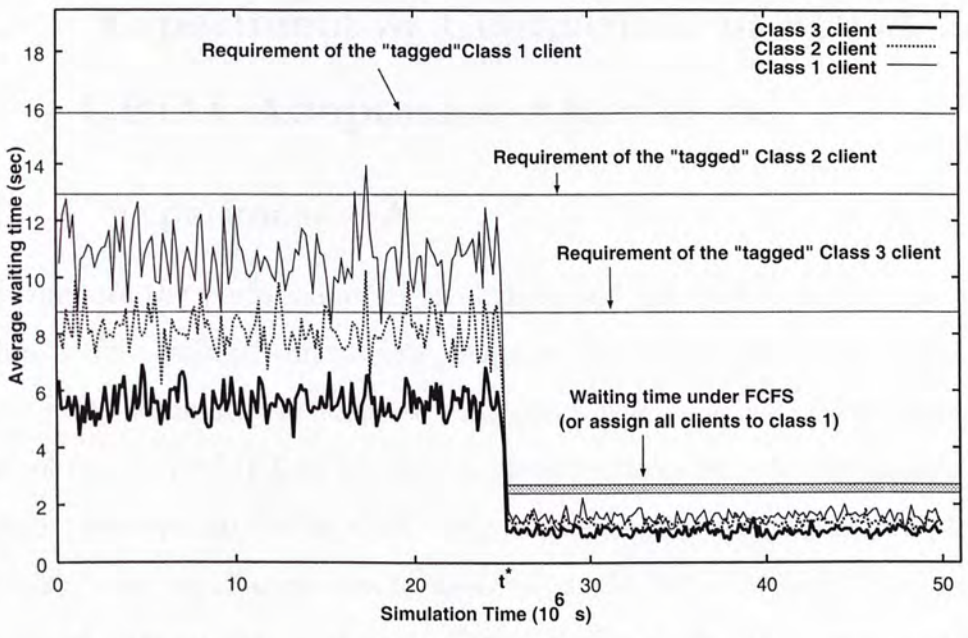


Figure 5.2: Waiting times for 3 different clients wherein the system workload is reduced by 30% at $t = 25 * 10^6$ seconds

Another scenario we consider is that after the admission process, all admitted clients submit requests based on their specified maximum arrival rates. However, at $t^* = 25 \times 10^6$ seconds, 30% the of admitted clients stop submitting requests to the web server. Those *"active"* clients still submit requests based on their specified maximum arrival rates. Figure 5.2 illustrates the waiting times of the three tagged clients from their respective classes. As we can observe, if all these three clients can mark their requests as class 1 after t^*, they will pay a much lower usage cost and their waiting time requirements can still be satisfied.

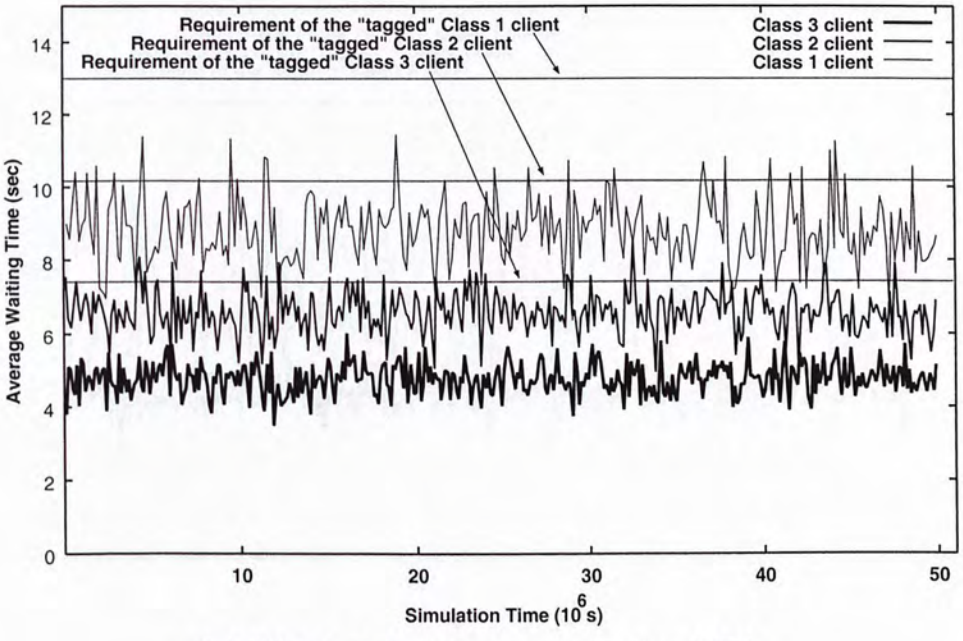
5.3 Experiment 3: Comparison of SBDA and CBDA Adaptation Algorithms

5.3.1 Experiment 3.A

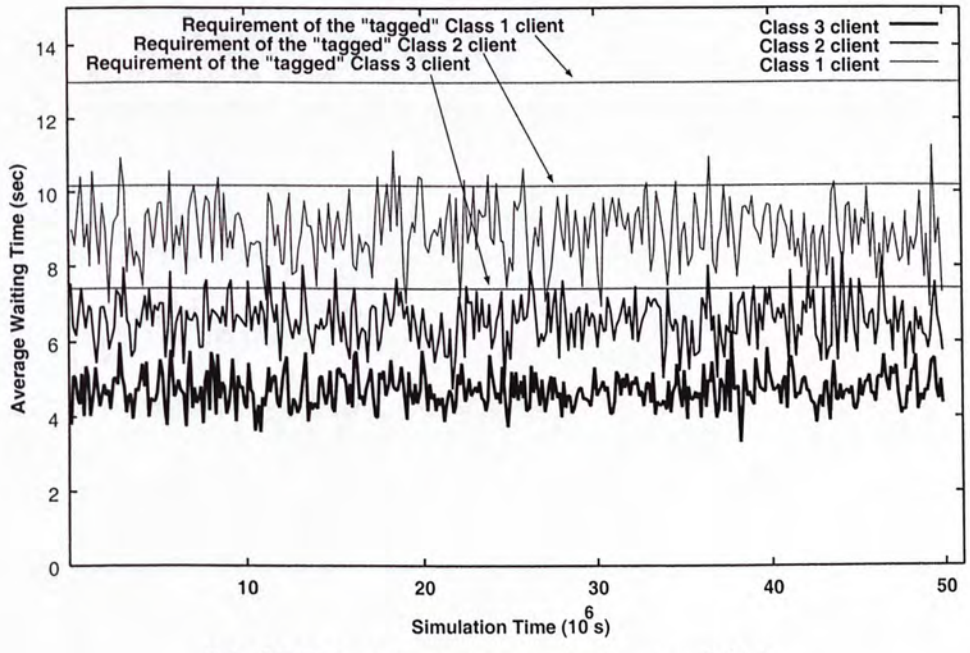
We compare the performance of the SBDA and the CBDA adaptation algorithms. The waiting time requirements of the clients are drawn uniformly from $[6, 20]$ seconds. The aggregate request rate is $\lambda_r = 1.2$. After admission control (by either MPA or MAA), we classify clients into $N = 3$ classes. We simulate the system for 50×10^6 seconds. During the simulation period, admitted clients can change class by using either the SBDA or CBDA algorithms described in the previous chapter. The arrival rate of each client can change during the simulation. Specifically, within a measurement period of length T, each client can change its arrival rate five times – with probability of 0.8 that the arrival rate is equal to the maximum arrival rate, with probability of 0.1 that the arrival rate is equal to 90% of the maximum arrival rate, and with probability of 0.1 that the arrival rate is equal to 80% of the maximum arrival rate. We consider a *"tagged"* client from *each* of the three classes and we plot their waiting times. Each point of the plot is the average waiting time of the previous 200 requests by the tagged client.

Figures 5.3 and 5.4 illustrate the waiting time of the three tagged clients under the SBDA and CBDA adaptation algorithms. The aggregate request rate λ_r (before admission control) is generated by a Poisson process with rate $\lambda_r = 1.2$. For Figure 5.3, we use MPA as the admission control algorithm at time $t = 0$. The three tagged clients have maximum average waiting time requirements of 12.996, 10.171, 7.404 seconds, respectively. For Figure 5.4, we use MAA as the admission control algorithm at time $t = 0$. The three tagged clients have maximum average waiting time requirements of 15.771, 12.988, 8.775 seconds, respectively. From these figures, we observe that both SBDA

and CBDA are very effective in adapting to the workload of the server. All three tagged clients achieve an average waiting time less than their maximum average waiting time requirements. However, note that since CBDA has a much lower computation complexity, it is the preferred algorithm.

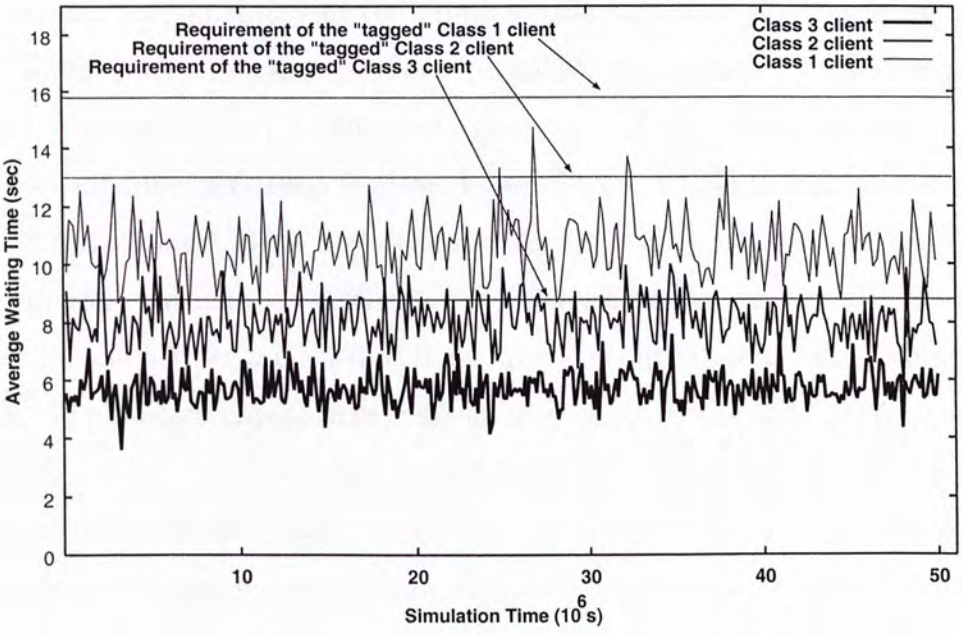


(a) SBDA: Admission control using MPA

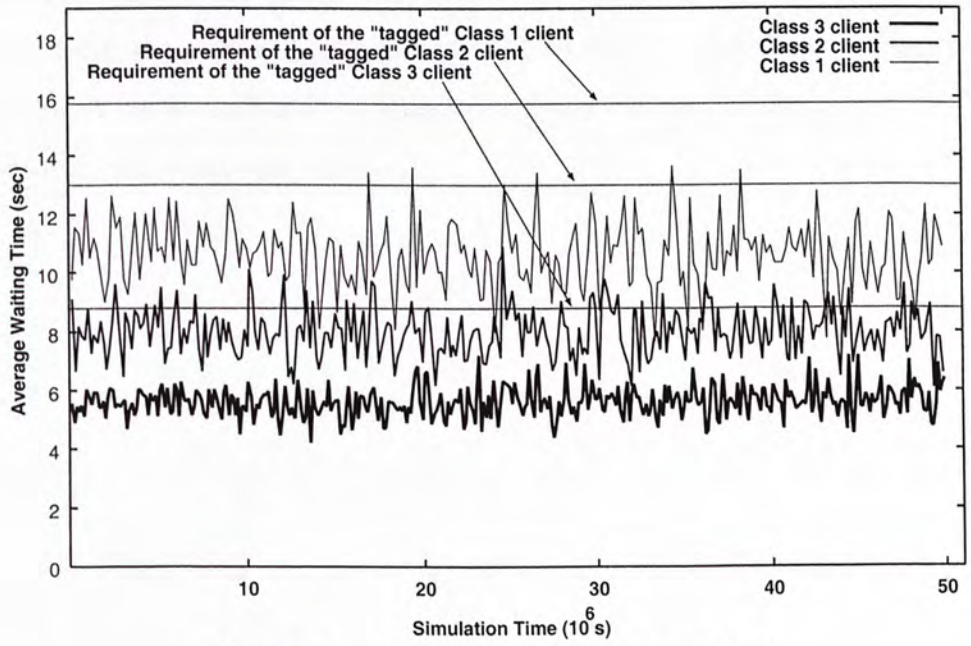


(b) CBDA: Admission control using MPA

Figure 5.3: Waiting times for three different clients under MPA admission control, Poisson arrival process.



(a) SBDA: Admission control using MAA

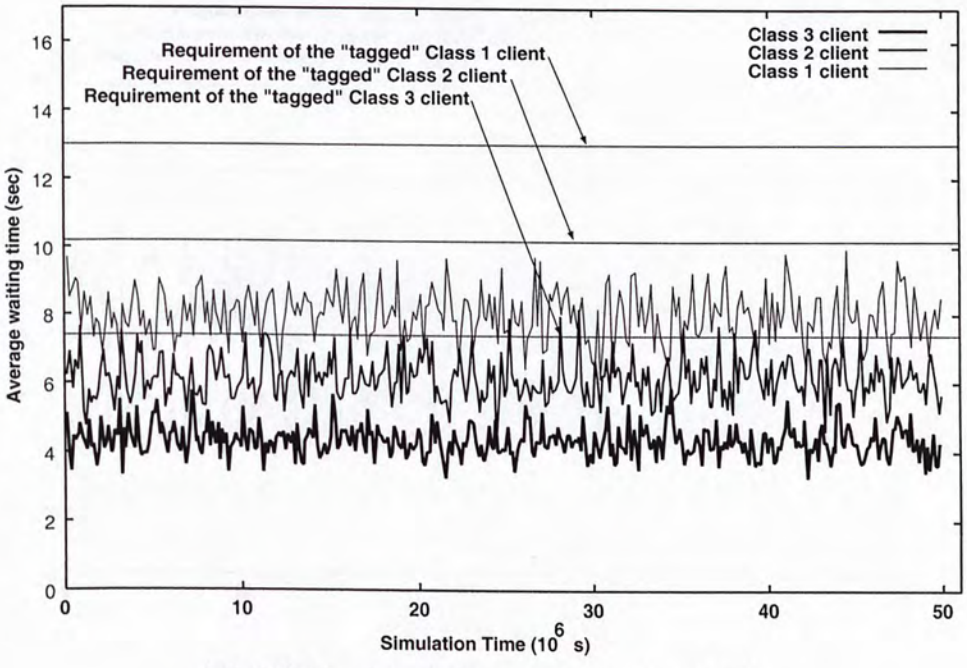


(b) CBDA: Admission control using MAA

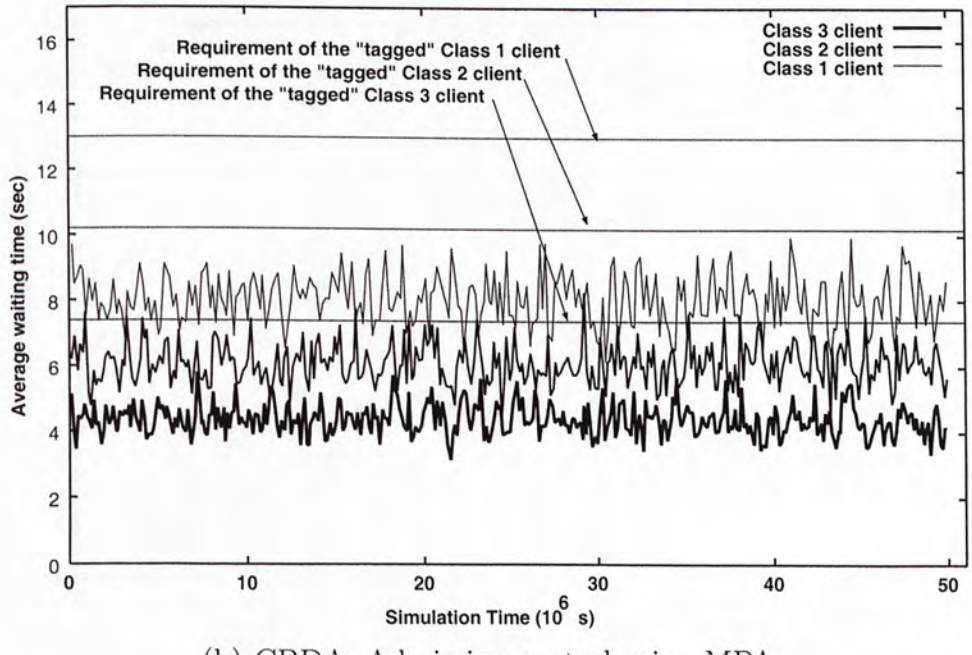
Figure 5.4: Waiting times for three different clients under MAA admission control, Poisson arrival process.

5.3.2 Experiment 3.B

We consider the capability of the proposed adaptation algorithms when the input traffic is *non-Poisson*. In this experiment, the aggregate request arrival rate has a mean of $\lambda_r = 1.2$. The traffic generation of each client is by Pareto or a Markov-modulated Poisson Process. Figure 5.5 and 5.6 illustrates the waiting time of SBDA and CBDA algorithms under Pareto, and Figure 5.7 and 5.8 illustrates the waiting time of SBDA and CBDA algorithms under MMPP. For Figure 5.5 and 5.7, we use MPA as the admission control algorithm at time $t = 0$. The three tagged clients have maximum average waiting time requirements of 12.996, 10.171, 7.404 seconds, respectively. For Figure 5.6 and 5.8, we use MAA as the admission control algorithm at time $t = 0$. The three tagged clients have maximum average waiting time requirements of 15.771, 12.988, 8.775 seconds, respectively. From these figures, we observe that both the SBDA and CBDA can adapt to the workload and their waiting time averages are less than their maximum average waiting time requirements. Note that, since CBDA has a much lower computational complexity, we should use CBDA to perform end point adaptation.

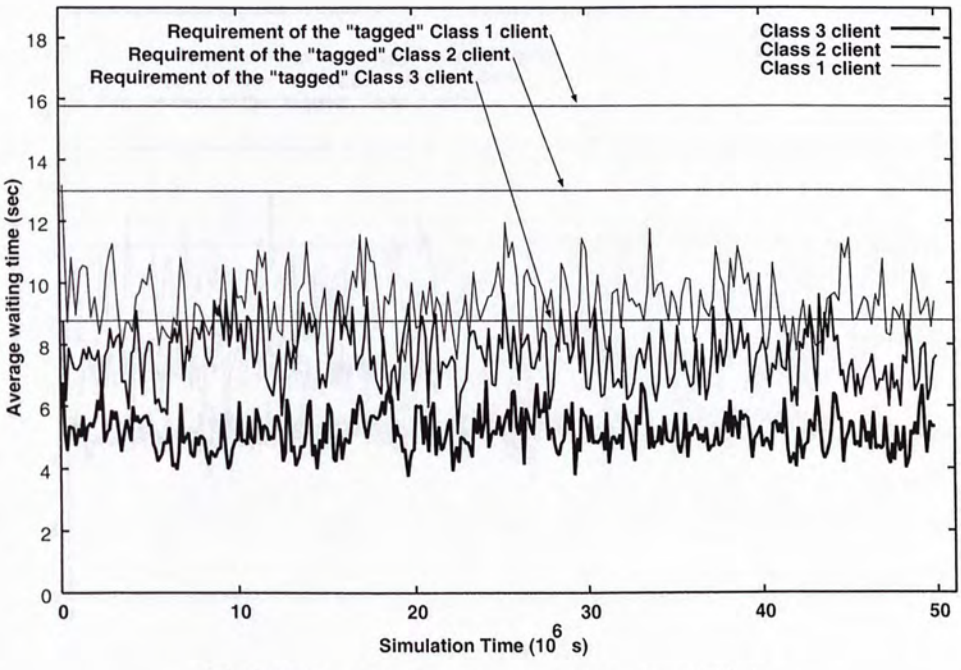


(a) SBDA: Admission control using MPA

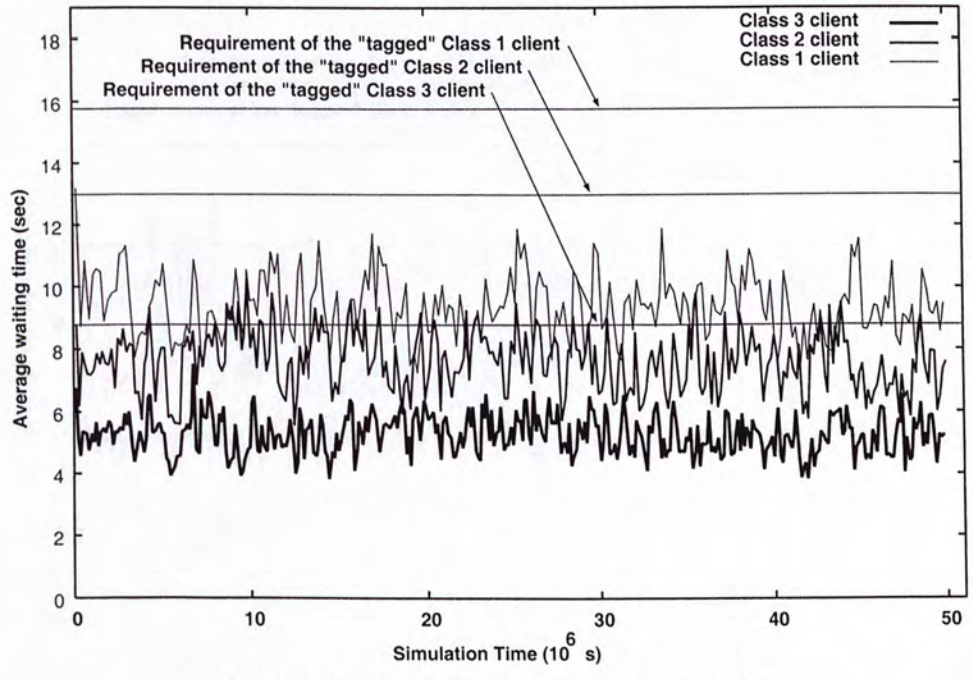


(b) CBDA: Admission control using MPA

Figure 5.5: Waiting time for three different clients under MPA admission control, Pareto arrival process.

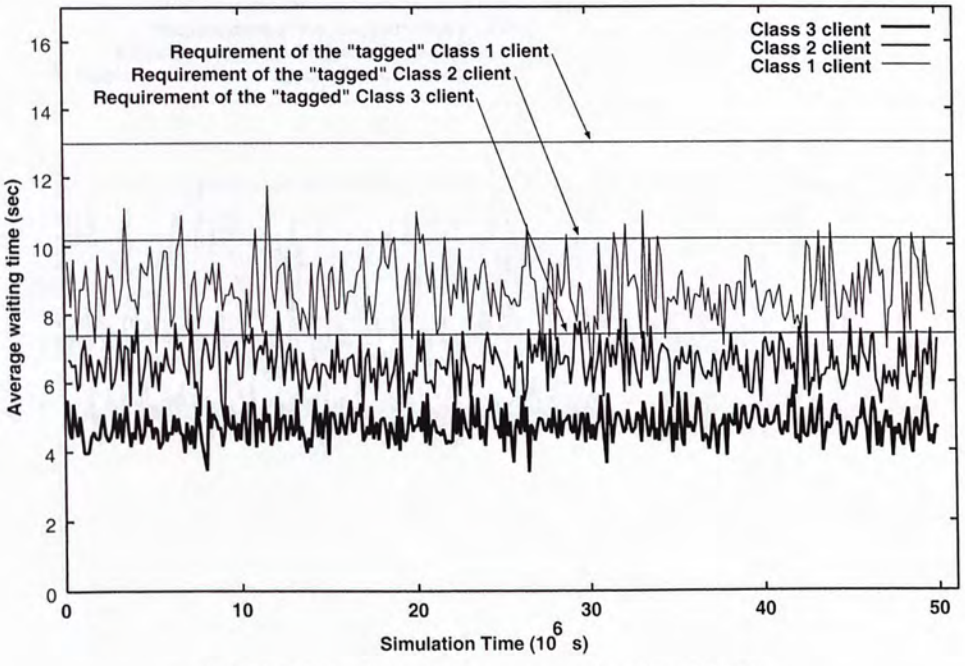


(a) SBDA: Admission control using MAA

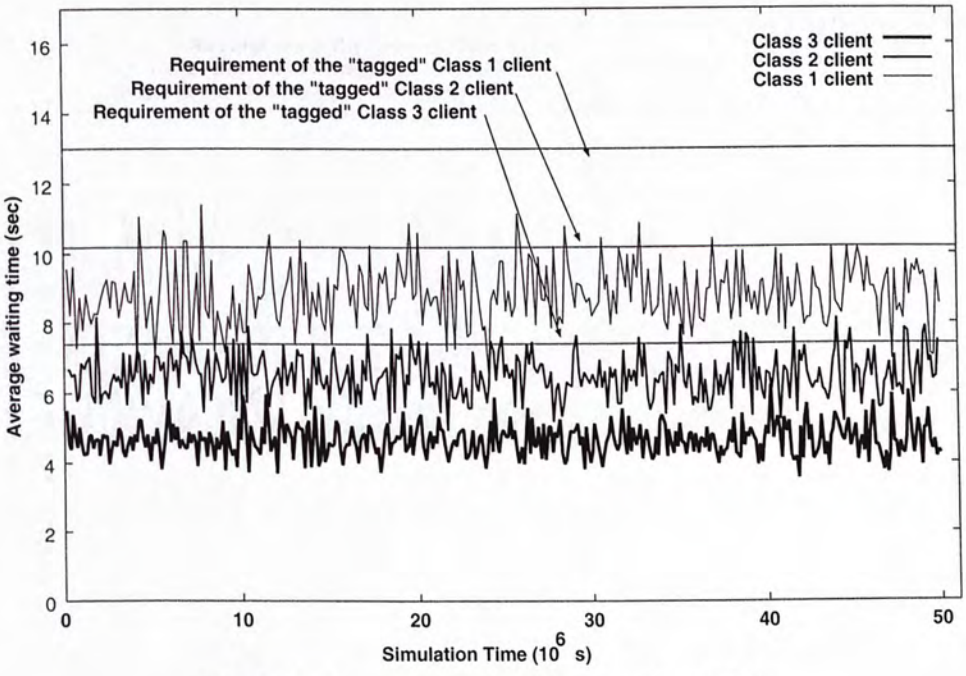


(b) CBDA: Admission control using MAA

Figure 5.6: Waiting time for three different clients under MAA admission control, Pareto arrival process.

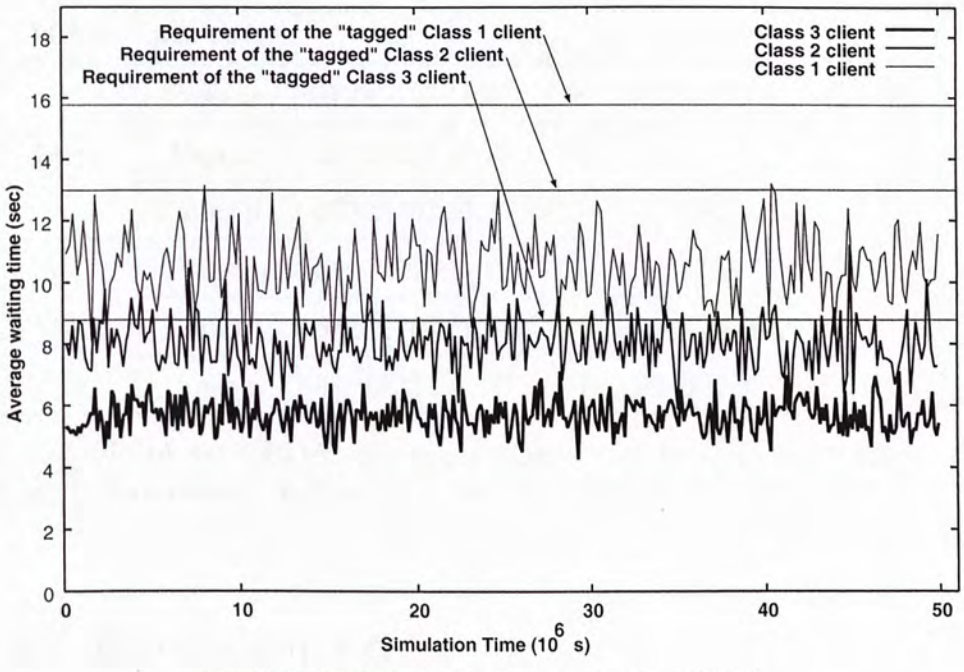


(a) SBDA: Admission control using MPA

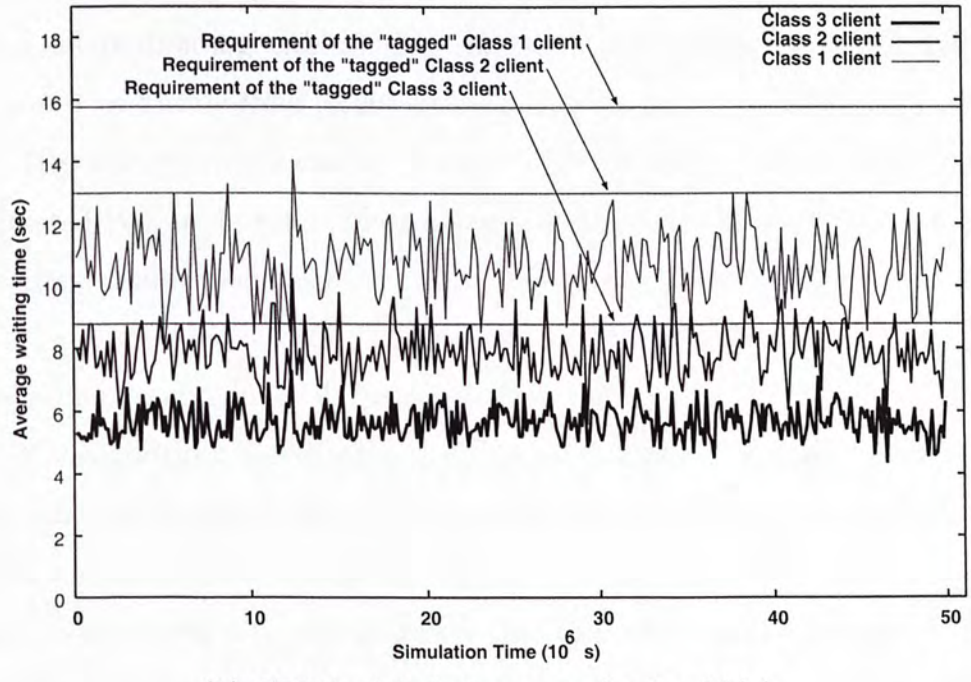


(b) CBDA: Admission control using MPA

Figure 5.7: Waiting time for three different clients under MPA admission control, MMPP arrival process.



(a) SBDA: Admission control using MAA



(b) CBDA: Admission control using MAA

Figure 5.8: Waiting time for three different clients under MAA admission control, MMPP arrival process.

MPA/ MAA	**arrival process**	SDBA	CBDA
MPA	Poisson	67381411 (1.37, 1.38)	67520157 (1.39, 1.39)
	Pareto	69796511 (1.35, 1.37)	69924606 (1.39, 1.38)
	mmpp	67390287 (1.37, 1.37)	67669302 (1.39, 1.39)
MAA	Poisson	61979953 (1.36, 1.38)	62119948 (1.39, 1.39)
	Pareto	64302708 (1.35, 1.38)	64316810 (1.38, 1.38)
	mmpp	61948317 (1.37, 1.37)	62060446 (1.39, 1.39)

Table 5.7: SDBA vs. CBDA: aggregate utility of all transmitted packets. The numbers in parenthesis indicate the two achieved differentiation ratios.

5.3.3 Experiment 3.C

We compare the aggregate utility of all the admitted requests for the SBDA and CBDA adaptation algorithms. The waiting time requirements of the clients are drawn uniformly from $[6, 20]$ seconds. The aggregate request rate is $\lambda_r = 1.0$. The arrival process can be generated by a Poisson, Pareto, or Markov-modulated Poisson process. The desired differentiation ratios are $r_{1,2} = r_{2,3} = 1.4$. After admission control (by either MPA or MAA), we classify clients into $N = 3$ classes. We simulate the system for 50×10^6 seconds. During the simulation period, admitted clients can change class by using either the SBDA or CBDA algorithms described in the previous chapter. The arrival rate of each client can change during the simulation. Each client has the same arrival rate within a measurement period of length T. At the end of each measurement period, each client will change the arrival rate with probability of 0.8 that the arrival rate is equal to the maximum arrival rate, with probability of 0.1 that the arrival rate is equal to 90% of the maximum arrival rate, and with probability of 0.1 that the arrival rate is equal to 80% of the maximum arrival rate.

Table 5.7 illustrates the aggregate utility of transmitted packets under the SBDA and CBDA adaptation algorithms with different arrival processes. Irrespective of the adaptation algorithm (MPA or MAA) and the arrival process (Poisson, Pareto, or MMPP), CBDA usually obtains a larger aggregate utility than SDBA. The achieved differentiation ratios $r_{i,i+1}$ are also nearer to the desired ratios. Once again, since the CBDA algorithm has a much lower computational complexity as compared to the SBDA algorithm, we should use CBDA for performing the end point adaptation.

Chapter 6

Related Work

We briefly summarize related research. Recently, various authors have suggested that it is important to consider differentiated services for web servers [7, 8, 9] in order to complement the Internet differentiated services model. In [7], the authors propose a centralized algorithm to perform server partitioning so as to provide differentiated services. In [8], the authors propose to use the shortest-connection-first algorithm. Differentiation is made for short and long connections. Using their algorithm, short connections have a significant performance gain while long connections pay relatively little penalty. In [9], the authors consider a server that provides prioritized service to different classes of users. In [10], the authors consider a web service which provides bounded latency for different classes of requests. In particular, the authors consider isolation among service classes as well as session control to protect classes from performance degradation due to overload. The latency requirements and service model considered in [10] are not PDDS but it is interesting to see how one can incorporate the proposed algorithms into our work. Lastly, the authors in [11] propose a method to select classes under PDDS so that requests can achieve an absolute QoS measure. The major differences between our work and [11] are: (1) we provide admission control so that we can guarantee the QoS requirements of all admitted clients, and (2) our class selection algorithms (MPA and MAA) have lower (polynomial time) computational complexity.

Conclusion

We have considered a web server that can provide proportional-delay differentiated services. The advantage of this type of service is that the operator of the web server can provide a *fixed* and *pre-specified* performance spacings between different classes of requests. Based on the performance spacings, the operator can legitimately charge a higher usage cost for clients in a higher service class. Each client has a maximum average waiting time QoS requirement. We prove that the general assignment problem is NP-complete. We present two efficient admission control algorithms that either maximize the potential profit or maximize the number of admitted clients into the system. We show that these admission control algorithms are computationally efficient and at the same time, the resulting class vector is a minimum feasible admitted class vector. To further reduce the usage cost, we also present two end point adaptation algorithms. One is server-based while the other is distributed. The distributed approach is based on a non-cooperative game technique. We show that the distributed approach has lower computational cost and can dynamically adapt to the server's workload. We also carry out experiments to illustrate the effectiveness of these algorithms under different parameters. For example, different traffic generation processes (Poisson, MMPP, or Pareto), different waiting time ratio specifications and different utility functions.

Bibliography

[1] C. Dovrolis, D. Stiliadis, and P. Ramanathan, Proportional differentiated services: Delay differentiation and packet scheduling, in *ACM SIGCOMM'99*, pages 109–119, 1999.

[2] L. Kleinrock, *Queueing Systems: Vol 2.*, Wiley-interscience, New York, 1976.

[3] M. Leung, J. C. Lui, and D. K. Yau, Characterization and performance evaluation for proportional delay differentiated services, in *International Conference on Network Protocols*, pages 295–304, 2000.

[4] M. Leung, J. C. Lui, and D. K. Yau, IEEE/ACM Transactions on Networking, 9(6) (2001).

[5] S. Shenker., IEEE Journal of Selected areas in Communication **13**, 1141 (1995).

[6] R. Gibbons, *Game Theory for Applied Economists.*, Princeton University Press, 1992.

[7] V. Cardellini, E. Casalicchico, M. Colajanni, and M. Mambelli, Web switch support for differentiated services, in *Performance and Architecture of Web Servers (PAWS), Boston*, 2001.

[8] M. Crovella, R. Frangioso, and M. Harchol-Balter, Connection scheduling in web servers, in *Proceedings of USITS'99, Boulder*, 1999.

[9] H. Zhu, H. Tang, and T.Yang, Demand-driven service differentiation in cluster-based network servers, in *Proc. IEEE Infocom 2001, Anchorage, Alaska*, 2001.

[10] V. Kanodia and E. Knightly, Multi-class latency-bounded web services, in *IEEE/IFIP IWQoS 2000, Pittsburgh, PA*, 2000.

[11] C. Dovrolis and P. Ramanathan, Proceedings of the 2001 IEEE International Conference on Network Protocols (2001).

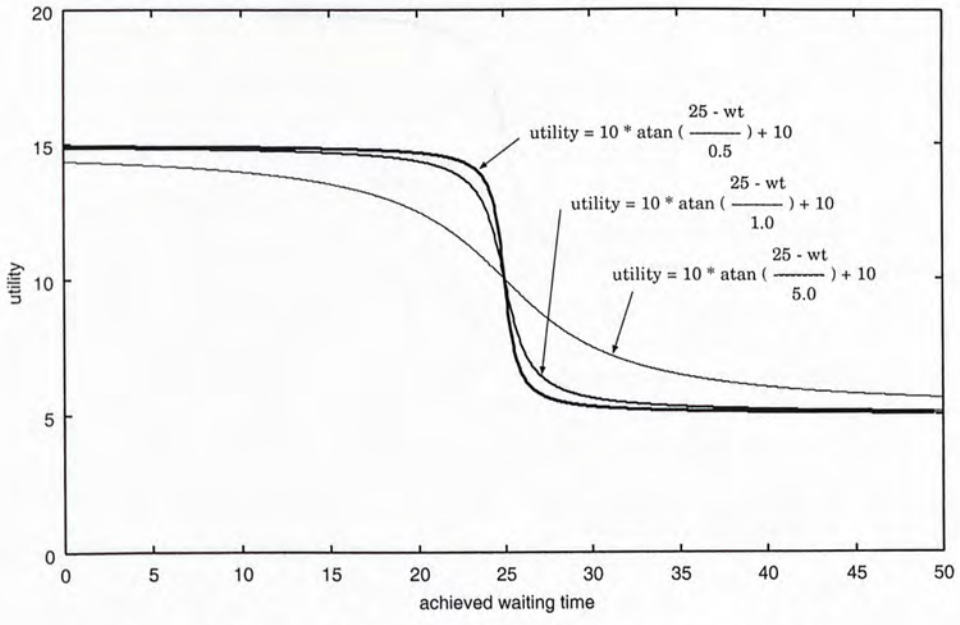


Figure 1: Utility functions with different values of A.

Appendix A

In general, the utility function in Figure 2.2 is a non-increasing function. It has a maximum and a minimum value indicating the maximum and minimum utility the client gets when the client receives a service of zero and infinity waiting time, respectively. The slope of the utility function represents the client's sensitivity to the waiting time. The steeper the slope, the more sensitive it is to the achieved waiting time. There is an inflection point in the utility function, which indicates the client has a maximum average waiting time requirement. To have the above properties, we use an arc tangent function to represent the client's utility function. i.e.

$$U(A, B, C, D, wt) \quad = \quad D \times \tan^{-1}(\frac{C - wt}{A}) + B$$

where A is the client's general sensitivity to the achieved average waiting time, B is the average utility of the client, C is the maximum average waiting time requirement, D is the maximum increase of the client's utility, and wt is the achieved waiting time of the client.

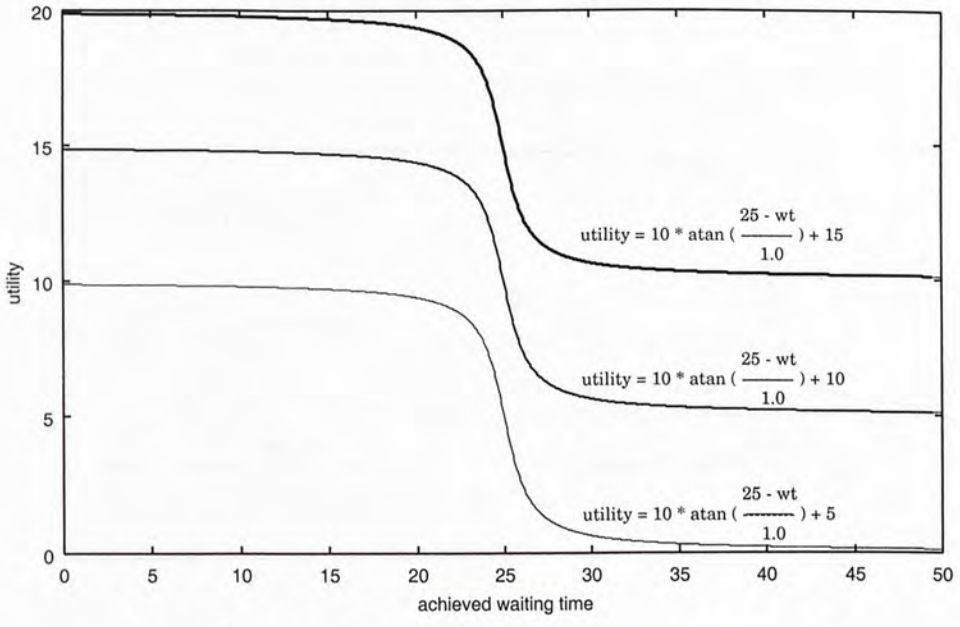


Figure 2: Utility functions with different values of B.

Utility templates U1, U2, and U3 have the same B, C and D parameter values. Parameter B is set to be half the value of D, so that a client always gets a non-negative utility. Parameter C is a random variable within the range of waiting time requirements as stated in the experiments. Parameter D is a random variable uniformly distributed in $[2, 3]$. Parameter A is a random variable in $[0.1, 0.3]$, $[0.2, 0.6]$, and $[0.4, 1.2]$ for templates U1, U2, and U3, respectively. The different values of parameter A for U1, U2, and U3 test the effect of the clients' sensitivity to the waiting time requirement on system efficacy. Utility template U4 is nearly the same as U1. The only difference is in parameter D, which is in $[2, 3]$ for U1 and in $[1, 3]$ for U4. This is to test the effect of maximum increase of clients' utility on system efficacy.

Figure 1 illustrates the effect of value A on the utility. When A increases, the sensitivity of the client to the waiting time decreases. Figure 2 illustrates the effect of value B on the utility. When B increases, the average utility of the client increases given same delay. Figure 3 illustrates the effect of value C on the utility. When C increases, the delay requirement of the client increases.

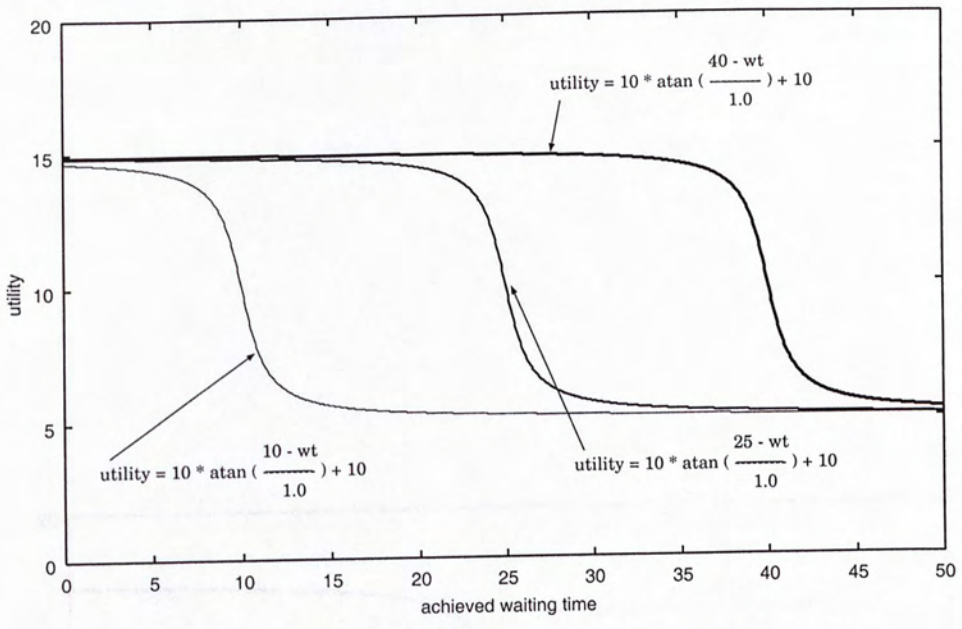


Figure 3: Utility functions with different values of C.

Figure 4 illustrates the effect of value D on the utility. When D increases, the maximum increase of utility of the client increases.

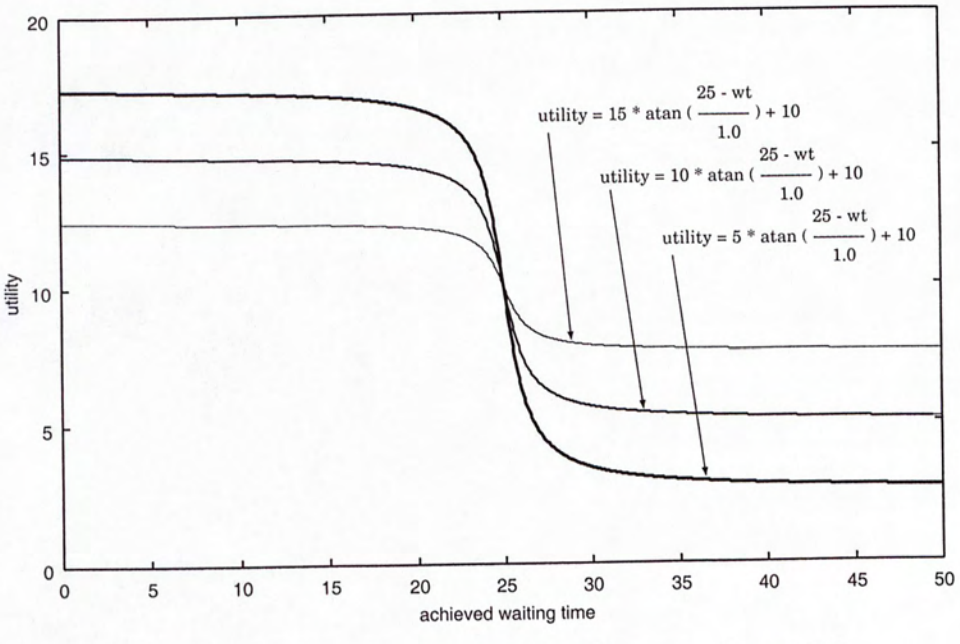


Figure 4: Utility functions with different values of D.